Stefan Schildbach

Aufbau eines Servers für die Energieprofil-basierte Suche

eingereicht als

Bachelorarbeit

an der

HOCHSCHULE MITTWEIDA

UNIVERSITY OF APPLIED SCIENCES

Mathematik / Naturwissenschaften / Informatik

Mittweida, 2010

Erstprüfer: Prof. Dr. rer. nat. Dirk Labudde

Zweitprüfer: Dipl.-Inf. (FH) Daniel Stockmann

Vorgelegte Arbeit wurde verteidigt am: 09.09.2010

Bibliographische Beschreibung:

Schildbach, Stefan:

Aufbau eines Servers für die Energieprofilbasierte Suche. – 2010. – 53 S.

Mittweida, Hochschule Mittweida,

Fakultät Mathematik / Naturwissenschaften / Informatik,

Fachbereich Informatik,

Bachelorarbeit, 2010

Kurzreferat:

Ziel der Bachelorarbeit ist es, eine Webanwendung für den biotechnischen Bereich der Hochschule Mittweida zu entwickeln. Dabei sollen errungene Kenntnisse und Tools aus der bioinformatischen Bibliothek für Forschung und Lehre bereit gestellt werden. Der Aufbau eines Servers mit einer nutzerfreundlichen Oberfläche steht dabei im Vordergrund.

Unter Berücksichtigung dieser Zielstellung werden zuerst die benötigten Tools aufbereitet und für die Integration in die Anwendung angepasst. Danach wird ein passendes Interface für die Dateneingabe erzeugt. Zum Schluss werden die Ausgaben der Tools für den Nutzer aufbereitet und dargestellt.

Inhaltsverzeichnis

1. Einleitung

1.1. Grundlegendes aus der Biotechnologie

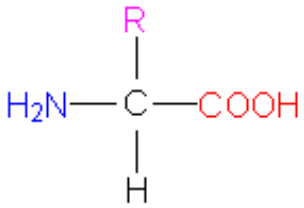
1.1.1. Aminosäuren

Der Begriff Aminosäuren wird häufig vereinfachend als Synonym für die proteinogenen Aminosäuren verwendet. Diese α-Aminosäuren sind die Bausteine der Proteine. Es existieren auch so genannte nicht-proteinogene natürlich vorkommende Aminosäuren, die biologische Funktionen haben, aber nicht am Aufbau von Proteinen beteiligt sind. Im Folgenden wird immer von proteinogenen Aminosäuren gesprochen.

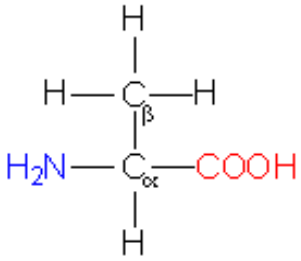
Aminosäuren sind eine Klasse organischer Verbindungen mit mindestens einer Carboxygruppe (–COOH) und einer Aminogruppe (–H2N), sowie einem Rest (R). Dieser Aufbau ist bei allen Aminosäuren gleich. Unterscheidbar werden sie durch die Struktur, die den Rest der Aminosäure bildet.

Das Kohlenstoffatom (C), welches die Carboxygruppe, die Aminogruppe und den Rest verbindet wird als Cα-Atom bezeichnet. Das mit diesem verbundene Kohlenstoffatom aus der „Restgruppe" bezeichnet man als Cβ-Atom.

Mehrere miteinander verbundene Aminosäuren bilden eine Sequenz.



Aufbau einer Aminosäure

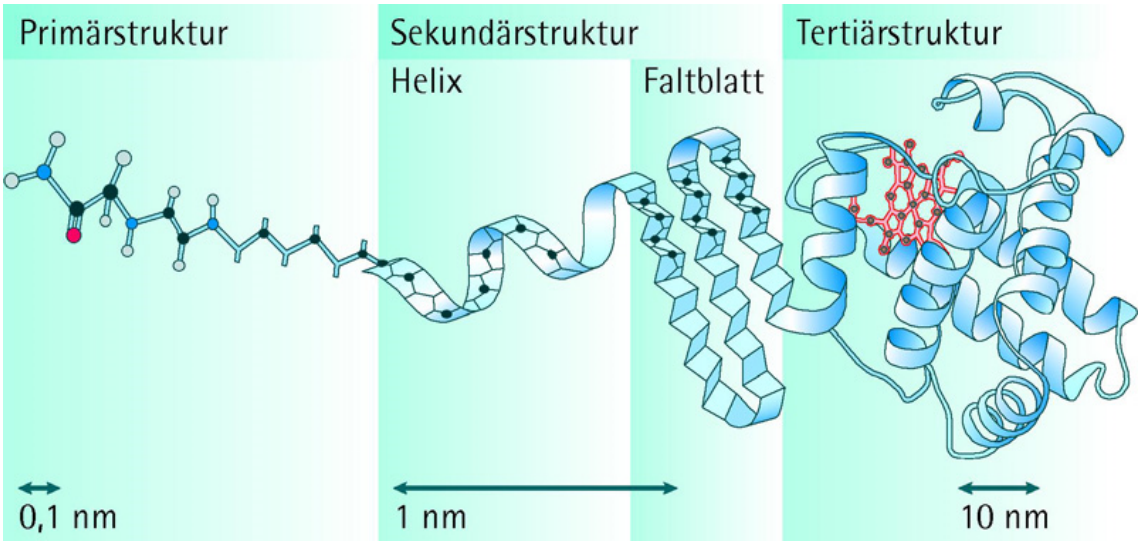


Aufbau einer Aminosäure am Beispiel von Alanin

1.1.2. Proteine

Proteine sind existenziell für jeden Organismus. Sie variieren stark in ihren Funktionen und sind an allen Stoffwechselvorgängen in Lebewesen beteiligt. In diesem Zusammenhang werden sie in zwei große Gruppen eingeteilt. Die Erste und weitaus Größere bilden die globulären (nicht-transmembrane) Proteine. Sie besitzen einen großen Wirkungsbereich, beispielsweise als Enzyme zur Katalyse chemischer Reaktionen oder als Antikörper zur Abwehr krankheitsauslösender Erreger. Membranproteine, als zweite Gruppierung, kommen wesentlich seltener vor. Sie sind an Zellmembranen angelagert oder auch eingelagert und werden dann als Transmembranproteine (transmembrane Proteine) bezeichnet. Sie geben den Zellen Stabilität und sorgen für den Stoffaustausch mit umgebenden Zellen. Die Grundeinheiten aller Proteine bilden die Aminosäuren. Diese werden bei der Proteinbiosynthese an den Ribosomen zu der Primärstruktur eines Proteins verknüpft. Das Ausbilden von Helices, Faltblättern oder Loops wird als Sekundärstruktur bezeichnet. Diese werden über Wasserstoffbrückenbindungen stabilisiert. Die Tertiärstruktur kennzeichnet die räumliche Anordnung der Aminosäuren zu einer Polypeptidkette. In der Tertiärstruktur werden je nach Protein und dessen Sequenz verschiedene Motive ausgeprägt. Dazu gehören die Sequenzmotive sowie die Struktur- bzw.

3D-Motive zur Stabilisierung der Polypeptidkette. Die Aggregation mehrerer Polypeptidketten wird als Quartärstruktur bezeichnet.



Der strukturelle Weg eines Proteins von der Primär- bis zur Tertiärstruktur

1.1.3. PDB

Die PDB (Protein Data Bank) ist eine Datenbank experimentell bestimmter 3D-Strukturen biologischer Makromoleküle, wie zum Beispiel Proteinen und Nukleinsäuren. Sie dient als weltweite Anlaufstelle für Forscher und Studierende. Die Dateien, die in dieser Datenbank abgelegt wurden, beinhalten Daten zu dem Experiment und Atomkoordinaten der jeweiligen Moleküle. Zusätzlich dazu sind weitere Informationen, wie die Namen der Moleküle, Strukturinformationen, Sequenzen, das Vorhandensein von Liganden und noch einiges mehr aufgeführt. Die Oberhand über die Datenbank hält eine Organisation mit dem Namen wwPDB (Worldwide Protein Data Bank).

1.2. Grundlegendes aus der Informatik

1.2.1. Java

Die Programmiersprache Java wurde von der Firma Sun-Microsystems zunächst für den Internet-Bereich entwickelt. Java wird inzwischen jedoch als universelle und plattformunabhängige Programmiersprache eingesetzt. Sie kann zwischen Wertdatentypen und Referenzdatentypen unterscheiden und verfügt über eine automatische Speicherverwaltung. Alle Programmobjekte werden in Klassen definiert. Diese Klassen stehen in einer Hierarchie und können mittels einfacher Vererbung die Methoden ihrer Oberklassen nutzen. Als einzigartige Oberklasse wird die Klasse „Object" betrachtet, welche als einzige von keiner anderen Klasse erbt. Die Portabilität von Java wird dadurch erreicht, dass der Java-Compiler keinen direkt ausführbaren Maschinencode erzeugt, sondern einen architekturneutralen Bytecode. Dieser Bytecode wird bei der Programmausführung von einer Java-VM interpretiert oder während der Ausführung in Maschinencode übersetzt. Die Abhängigkeit vom unterliegenden Betriebssystem ist in Java durch standardisierte Programmbibliotheken weitestgehend ausgeschaltet. Die in Java eingebauten Sicherheitsmechanismen (Sandbox) ermöglichen es relativ risikofrei über das Netz geladene Java-Programme (Java-Applets) auf dem lokalen Rechner auszuführen.

1.2.2. Web-Server

Das World Wide Web hat sich von einem System zum Abruf und zur Darstellung von statischen Inhalten zu einem komplexen System von dynamisch erzeugten Seiten, die sich den Benutzeranforderungen anpassen, entwickelt. Die dynamischen Inhalte auf den Webseiten können clientseitig (z.B.: JavaScript, Java-Applets) oder auch serverseitig (z.B.: Java Servlets, Java Server Pages) erzeugt werden.

Diese Weiterentwicklung zu einem dynamischen System stellt den Web-Browser als universelle Schnittstelle für verteilte Anwendungen in ein neues Licht. Damit ist nun keine Installation von Software mehr notwendig um auf eine bestimmte Anwendung zuzugreifen. Dabei werden die benötigten Eingabedaten über den Web-Browser an einen Web-Server geschickt, welcher diese dann weiter verarbeitet und anschließend die Ergebnisse zurück an den Web-Browser schickt.

Der Web-Server stellt dabei ein Programmsystem dar, welches die intern gespeicherten Anwendungen nutzt und anschließend die erzeugten Ergebnisse durch andere Internet-Hosts abrufbar macht. Ein solches System bieten zum Beispiel die Java Servlets und die Java Server Pages.

1.2.3. Java Servlets

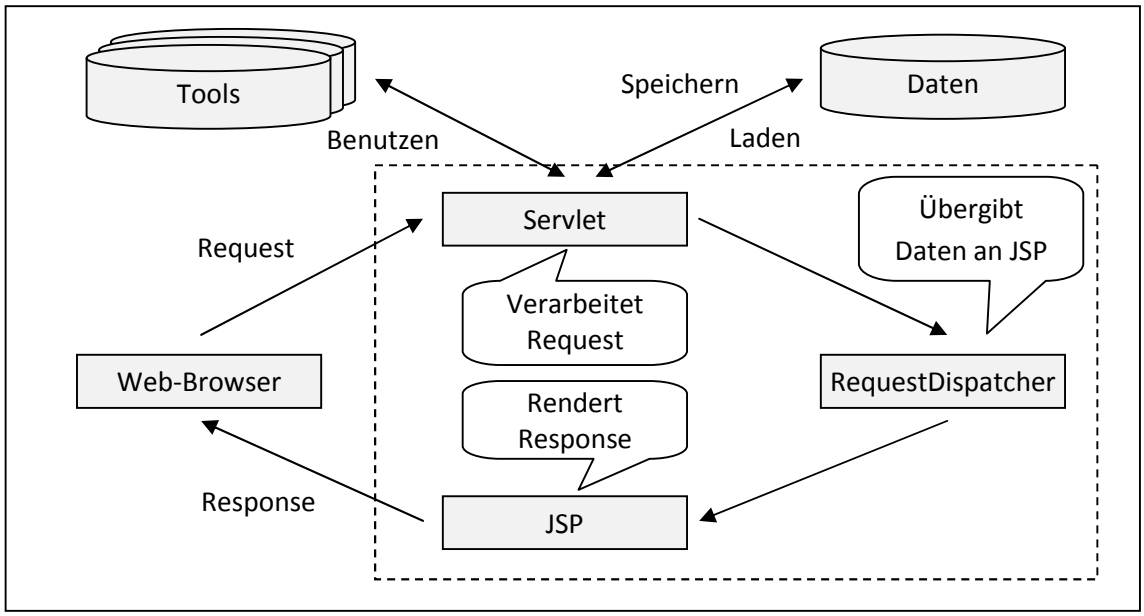
Java Servlets sind keine eigenständigen Java-Programme und auch keine clientgesteuerten Java-Applets. In der Regel sind es von der Klasse HttpServlet abgeleitete Klassen, die die Methoden doGet, doPost oder auch beide überschreiben. Das Servlet kann in Form einer URL vom Client angesprochen werden. Geschieht dies über einen HTTP-GET-Request, wird die Methode doGet ausgeführt. Wird das Servlet über einen HTTP-POST-Request angesprochen, wird die Methode doPost ausgeführt. Diese Methoden stellen die übergebenen Parameter in Form des Zugriffs auf den Header- und Body-Bereich des HTTP-Requests, sowie auch die Kontrolle über die Ausgabe in Form des Zugriffs auf die Header- und Body-Daten des HTTP-Response zur Verfügung.

Wird ein Servlet angesprochen, so wird nicht jedesmal ein Objekt der Servlet-Klasse erzeugt, sondern der unterliegende Web-Server (z.B.: Apache Tomcat) erzeugt einmalig ein solches Objekt. Dies geschieht nicht bei jedem Aufruf, sondern einmalig beim ersten Aufruf des Servlets oder beim Hochfahren des Servers. Alle einkommenden Requests werden dann in einem eigenen Thread abgearbeitet. Somit greifen alle Benutzer auf dasselbe Servlet-Objekt zu. So müssen bei Anwendungen, bei denen für jeden Benutzer unterschiedliche Daten verarbeitet werden spezielle Maßnahmen zum Session-Tracking eingesetzt werden.

1.2.4. Java Server Pages

Servlets enthalten HTML-Text in Ausgabe-Anweisungen innerhalb eines Java-Programmes um ihre Ergebnisse zu präsentieren. Bei Java Server Pages (JSP) ist es genau umgekehrt. JSP-Seiten bestehen aus HTML-Text, in denen Java-Anweisungen eingebettet werden können. Dabei sind Servlets und JSPs nicht grundlegend verschieden, denn JSPs werden zur Laufzeit vom Web-Server in Servlets umgewandelt. Somit unterschieden sich beide nur in der Notation und der grundlegenden Absicht des Programmierers. Servlets sind dabei besser geeignet, wenn

viel Java-Code ohne größere Visualisierung benötigt wird. JSPs hingegen dienen als Interface für die Benutzerkommunikation ohne größere Berechnungen.



Zusammenspiel von Browser, Servlet, Tools und Daten in „eProS"

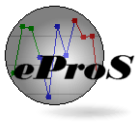
2. Grundlagen

2.1. Die Namensgebung

Der nahezu erste Schritt war die Namensgebung der Webanwendung, unter welchem das Projekt laufen soll. Relativ schnell wurde der Name „eProS" gewählt. Davon Abweichend und im Laufe der Entwicklung gab es noch einige Modifikationen, wie zum Beispiel „eProST" (Energy Profile Search Tool), aber letztendlich hat sich „eProS" als „Energyprofile server" durchgesetzt. Dies hat sich aus dem Trend der bioinformatischen Tools, die der Server anbieten soll ergeben, welche auch alle mit einem kleinen „e" und einer nachfolgenden Spezifikation bestehen. Dabei steht das führende kleine „e" immer für den Bezug auf die Energie.

2.2. Das Logo

Das Logo für „eProS" besteht aus einer Kugel mit einem innerhalb liegenden Liniendiagramm, welches verschiedene Farben aufweist. Leicht versetzt darüber ist der Schriftzug „eProS" zu lesen.



Dieses Diagramm ist eine Anlehnung an ein Energieprofil. Die verschiedenen Farben kennzeichnen dabei in dem Ausschnitt die Sekundärstrukturen. Die das Diagramm umgebende Kugel stellt die zur Berechnung der Energieprofile benötigte 8Å-Kugel dar. Somit bildet das Logo die Kernelemente hinter „eProS" ab.

2.3. Lokaler Testserver

2.3.1. Tomcat

Um die Funktionalität und die Arbeitsweise des Servers zu Testen wurde ein lokaler Testserver installiert, bevor das System auf einen Produktionsserver ziehen sollte. Dafür angeboten hat sich der Apache Tomcat in der Version 6.0.26. Dieser war zum Zeitpunkt der Arbeit der aktuellste Release, welcher nicht in der Beta-Phase stand.

Um den Tomcat zu installieren, wurde nur ein einfaches Archiv heruntergeladen und entpackt. Es war nicht notwendig weitere Konfigurationen vorzunehmen, der entpackte Inhalt stellte den lauffähigen Web-Server dar. Da für den lokalen Testbetrieb keine Sicherheitseinstellungen und Benutzerverwaltungen nötig waren, wurde die Standardkonfiguration nicht weiter verändert, da sie allen gewünschten Anforderungen entsprach. Die Einrichtung des Servers war damit schon abgeschlossen.

Der Zugriff auf den Web-Server erfolgte über den Browser mittels der URL „http://localhost:8080". Damit war es nun möglich die eigentliche Anwendung zu entwickeln.

Die wichtigsten Bedienelemente waren zum einen die Batchdateien zum Starten (\bin\startup.bat) und Beenden (\bin\shutdown.bat) des Servers, welche ein kontrolliertes hoch- und runterfahren ermöglichten. Zum anderen ist es das Deploy-Verzeichnis, der Speicherort für die Webapplikationen (\webapps). In diesen Ordner müssen die gepackten WAR-Dateien (Web-Archiv) hinein kopiert werden. Tomcat erkennt neue oder auch veränderte Archive von selbst und beginnt diese zu entpacken und in die interne Struktur zu integrieren.

Falls eine ältere Version eines Archivs bereits im System vorhanden war, wird diese gelöscht und die neuere Version nimmt deren Platz ein.

2.3.2. Die WAR-Dateien

Die WAR-Dateien sind vollständige Webanwendungen, welche als JAR oder ZIP verpackt werden und die spezielle Endung „.war" erhalten. Durch das Deployment einer einzigen WAR-Datei kann eine gesamte Web-Anwendung auf den Server aufgespielt werden. Jede WAR-Datei muss einem bestimmten Aufbau folgen, damit sie korrekt integriert werden kann. Das Kernstück ist das „WEB-INF"-Verzeichnis. In diesem befindet sich der sogenannte Deployment Descriptor (web.xml). Dieser definiert alle Eigenschaften der Webanwendung, sowie alle ansprechbaren Servlets.

In einem Unterordner „WEB-INF\classes" werden alle kompilierten Java-Klassen (*.class-Dateien) abgelegt. Alle für die Webapplikation benötigten Bibliotheken können in den Ordner „WEB-INF\lib" eingeordnet werden. Auf die im Ordner „WEB-INF" enthaltenen Dateien kann von außerhalb, also vom Client aus, nicht zugegriffen werden. Man kann zwar JSP-Dateien mit in diesen Ordner ablegen, aber das ist nur sinnvoll, wenn diese von einem Servlet mittels Request-Forwarding aufgerufen werden. Damit kann man das versehentliche Aufrufen einer JSP-Seite ohne die benötigten Attribute vermeiden. Für Java Servlet Pages, die auch von außen angesprochen werden können, müssen die entsprechenden Dateien außerhalb des „WEB-INF"-Verzeichnisses liegen. So bietet es sich an, Bilder und andere Ressourcen, sowie Stylesheets, die von den ansprechbaren JSP-Seiten benötigt werden, auch außerhalb des „WEB-INF" zu speichern. Dabei ist es kein Problem, eine tiefere Ordnerstruktur anzulegen.

2.4. Produktionsserver

Der Umzug auf den Produktionsserver erfolgte in den letzten Entwicklungswochen, nachdem das System größtenteils funktionell und visuell ausgereift war. Damit ist ein weltweiter Zugriff auf die Anwendung über folgende URL möglich:

```
http://bioservices.hs-mittweida.de/Epros
```

Der Produktionsserver nutzt als Web-Server ebenfalls Tomcat, was die Übertragung der Daten mittels der WAR-Datei sehr komfortabel gestaltete. Der eigentliche Schwerpunkt des Umzugs lag in der Beschaffenheit der unterliegenden Betriebssysteme. Der lokale Testserver lief auf einer Windows-Plattform, der Produktionsserver hingegen befindet sich auf einer UNIX-Maschine. Damit mussten noch einige Anpassungen vorgenommen werden, vor allem in den Angaben der Pfade zu den verwendeten Daten.

2.5. Daten

2.5.1. Proteindatenbank-Dateien

Die PDB-Dateien (Proteindatenbank-Dateien) stellen den ersten großen Komplex der benötigten Daten dar. Mit Hilfe dieser PDBs können die 3D-Modelle visualisiert werden und sie bilden die Grundlage für die Berechnung der Energieprofile.

2.5.2. Energiedateien

Die Energiedateien gibt es in drei verschiedenen Ausführungen.

Die erste Variante ist die einfache Energieprofil-Datei (*.ep). Diese Datei ist eine Textdatei mit den primären Informationen über ein Energieprofil. Es werden der Name des Proteins, sowie die Informationen zu den einzelnen Aminosäuren abgelegt. Dazu zählen die Ketten-ID, der Name, die interne Nummer, die Sekundärstruktur und natürlich der Energiewert der Aminosäure.

Die zweite Variante stellen die detailierten Energieprofil-Dateien dar (*.ep2). Es sind ebenfalls Textdateien und sie erweitern die einfachen Energieprofil-Dateien um jeweils einen Zeilenbezeichner, sowie zwei weitere Informationen. Zum einen wird gespeichert, ob das vorliegende Energieprofil mit der Berechnung für transmembrane oder für nicht-transmembrane Proteine erstellt worden ist. Die zweite ergänzende Information ist ein Bereich mit Anmerkungen. In diesem werden Probleme oder Besonderheiten bei der Erstellung der Energiedatei vermerkt.

Die dritte Variante stellt die serialisierte Energiedatei (*.eps) dar. Diese beinhaltet alle Informationen, die beim Erstellen des Energieprofils zu vermerken waren. Angefangen vom Name des Proteins über die einzelnen Aminosäuren mit Raumkoordinaten und Energiewerten, bis hin zu den ergänzenden Anmerkungen. Diese Dateien werden intern verwendet um Berechnungen auszuführen und die 2D-Visualisierungen zu ermöglichen.

Im Folgenden wird auf diese dritte Variante der Energiedateien bezug genommen, wenn von Energiedateien (EPs) gesprochen wird.

2.5.3. ESR-Dateien

Die dritte große Gruppe von benötigten Dateien sind die ESR-Dateien. Diese bilden eine Anfrage an das Energie-Suchtool ab (energy search request-Dateien). Es handelt sich dabei um serialisierte Dateien, welche die Informationen der Anfrage speichern und in die auch das Resultat des Suchlaufs hineingeschrieben wird.

2.5.4. Datenbank oder flache Datenstruktur

Es wurde lange überlegt, ob der Einsatz einer Datenbank für die Datenhaltung relevant ist. Die zu speichernden Daten enthalten genug Informationen, dass man sie ohne weiteres in eine Datenbank einspeisen könnte und somit eine geordnete und saubere Datenmenge erhält, auf die zugegriffen werden kann.

Dennoch fiel die Entscheidung gegen den Einsatz einer Datenbank. Grund dafür war die Art der Benutzung der Daten. Diese werden immer spezifisch über ihre eindeutige ID, die PDB-ID angesprochen. Es ist nicht vorgesehen, nach spezifischen Eigenschaften der einzelnen Dateien zu filtern oder zu suchen. Somit bezieht sich der Zugriff auf die Daten auf eine einzelne Abfrage der ID. Der Einsatz einer Datenbank wurde damit als zu großer Overhead eher als störend empfunden, als ein hilfreiches Mittel zu sein. Um dennoch den Überblick zu behalten wird jede Datenart in einem separaten Ordner gehalten.

2.5.5. Speicherung

Das System ist derzeit so ausgelegt, dass neben der Webanwendung an sich ein weiterer Ordner in das Deployment-Verzeichnis des Web-Servers gelegt werden muss, der alle Daten beinhaltet. Damit wird erreicht, dass die Daten nicht jedes mal mit deployed bzw. hochgeladen werden müssen, wenn die Webanwendung aktualisiert wird. Dieses Verzeichnis muss den Namen „EprosData" tragen und beinhaltet 3 weitere Ordner.

- pdb – Ablageort für die PDB-Dateien
- ep – Speicherort für die EP-Dateien
- esr – hier befinden sich die ESR-Dateien

In den ersten beiden Ordnern liegt unmittelbar in erster Ebene ein vorgegebener Datensatz, der immer zum Abruf verfügbar ist. Dieser besteht derzeit aus etwa 4300 Dateien. Zusätzlich zu diesen Daten werden dynamisch Session-Ordner erzeugt. Das heißt, wenn ein Benutzer in seiner Session eine PDB-ID beantragt, welche nicht im Datensatz vorhanden ist, wird ein Ordner erstellt und die entsprechende PDB-Datei für die Dauer der Session heruntergeladen und das daraus berechnete EP gespeichert.

Der ESR-Ordner ist ähnlich aufgebaut, allerdings mit dem Unterschied, dass es hier keine vorgegebenen Datensätze gibt. Es wird also beim Erstellen einer Anfrage ein Verzeichnis mit der ID der Session des Benutzers angelegt und die entsprechenden Daten werden darin gespeichert. Weiterführend wird dieser Vorgang im Kapitel 5.8. - ESearchServlet beschrieben.

3. Nutzerinterface

3.1. Input

Dem Nutzer ist es möglich folgende Daten vorzugeben:

- Die PDB-ID des gewünschten Proteins [1) 2)]
- Eine eigene PDB-Datei hochladen [1) 2)]
- Eine eigene „*.ep"- oder „*.ep2"-Datei hochzuladen [1)]
- Die Eingabe einer Sequenz [2) 3)]

1) Es ist zusätzlich möglich, eine spezielle Kette auszuwählen, welche ausschließlich angezeigt werden soll

2) Es ist notwendig die Art der Berechnung vorzugeben. Dabei muss zwischen einer Berechnung von transmembranen und nicht-transmembranen Proteinen unterschieden werden.

Dabei stehen bei einigen Tools nicht alle Eingabemöglichkeiten zur Verfügung.

Weiterhin kann der Nutzer seine Anfrage mit folgenden Optionen spezifizieren, falls ein Alignment durchgeführt wird:

- Methode des Alignments (global oder lokales Alignment)
- Kosten für eine neue Lücke
- Kosten für das Erweitern einer Lücke

Für das Suchtool sind weiterhin folgende Optionen verfügbar:

- Anzahl der darzustellenden Ergebnisse
- Eingabe einer eMail-Adresse
- Eingabe einer Ergebnis-ID

Beim Absenden jeder Anfrage ist es möglich, den bisher angelegten temporären Session-Speicher zu löschen. Dies kann nützlich sein, wenn man mehrere verschiedene Dateien nutzen möchte, die das gleiche Protein beschreiben (die gleiche ID haben) und unter anderen Bedingungen aufgezeichnet wurden.

3.2. Navigation

Um im „eProS" zu navigieren gibt es eine Sidebar, die die Hauptnavigation ermöglicht. So können neben den Tools auch noch Zusätzliche Informationen zum Entwicklungsstand und den zu Grunde liegenden Statistiken abgerufen werden.

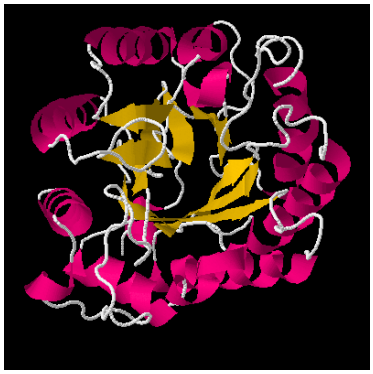
Die Tools selber besitzen noch einmal eine extra Tab-Navigation am oberen Rand. Damit ist es möglich zwischen den einzelnen Tools umher zu schalten und die Ergebnisse der jeweiligen Tools nicht zu verlieren.

Bei einem Klick auf einen Tab zeigt sich somit entweder die zuletzt berechnete Ausgabe oder die jeweilige Eingabemaske, falls das Tool noch nicht ausgeführt wurde. Auf jeder Seite gibt es im oberen Bereich die Möglichkeit zur Eingabemaske zurück zu wechseln.

3.3. Visualisierung

3.3.1. Jmol

Jmol ist ein Open-Source Visualisierungstool für 3D-Strukturen von Proteinen. Der Einsatz dieses Applets ermöglicht es mit Hilfe der PDB-Dateien, das gewünschte Protein räumlich darzustellen. Die Möglichkeiten der Interaktion beschränken sich auf das Drehen und Zoomen des Proteins, sowie das Markieren von einzelnen Aminosäuren. Die Einstellungen wurden so angepasst, dass alle weiteren Funktionalitäten des Applets abgeschalten wurden. Somit dient Jmol tatsächlich nur der 3D-Modellierung der Proteine um eine Vorstellung ihres Aussehens zu erhalten.

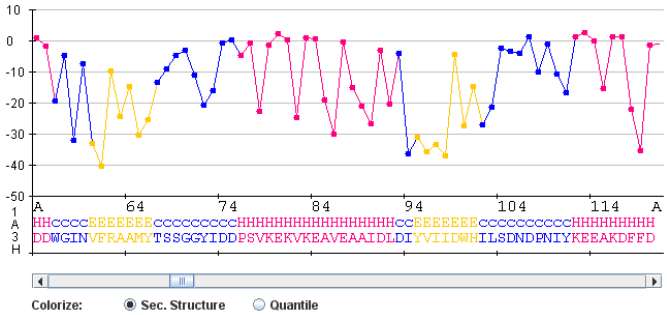


Das Protein 1A3H in Jmol visualisiert

3.3.2. eVis

„eVis" ist ein Applet zur Visualisierung von Energieprofilen. Die Energiewerte werden in einem einfachen Liniendiagramm dargestellt. Darunter wird die aktuelle Kette, die Sekundär-, sowie die Primärstruktur dargestellt. Es können auch zwei Energieprofile übereinander gelegt werden um Gemeinsamkeiten und Unterschiede festzustellen. Die Interaktionsmöglichkeiten für den Benutzer sind an zwei Stellen möglich. Einerseits können verschiedene Einfärbungen vorgenommen werden. Für mehrere Energieprofile kann man je einen Datensatz in einer Farbe anzeigen lassen. Weiterhin kann man nach Sekundärstrukturen, sowie nach Quantilen färben. Die zweite Interaktionsmöglichkeit ist das Markieren von Bereichen. So kann man verschiedene Bereiche mit der linken Maustaste hervorheben und mit der rechten Maustaste diese Markierungen wieder zurücksetzen. Wenn nur ein Datensatz dargestellt wird kann man im gesamten Diagramm markieren. Werden jedoch zwei Datensätze dargestellt ist es nur möglich im Datenbereich (Kette, Primär- und Sekundärstruktur) zu markieren.

Dabei werden sowohl die Einfärbungen als auch die Markierungen direkt in Jmol übernommen. Sie ermöglichen so den Übertrag der Energieprofile auf die räumliche Struktur der Proteine.



Das Energieprofil des Proteins 1A3H in „eVis" visualisiert

4. Zugrunde liegende Tools

4.1. eProfile

Das Tools eProfile (energy profile) legt den Grundstein für „eProS" und bildet das Herzstück der ganzen Anwendung. Es basiert auf dem von Dr. Frank Dressel im Zusammenhang mit seiner Dissertationsschrift entworfen Algorithmus zur Berechnung der freien Energie jeder Aminosäure, in einem nicht-transmembranen Protein. Der Algorithmus wurde so erweitert und angepasst, dass er nun auch auf transmembrane Proteine anwendbar ist und toleranter auf Fehler eingeht.

4.1.1. Nutzung

Um ein Energieprofil zu erzeugen, müssen folgende Schritte ausgeführt werden. Als Erstes muss natürlich eine Instanz der Klasse erzeugt werden:

```
EProfile eProfile = new EProfile();
```

Anschließend kann optional das Logging eingeschalten werden.

```
eProfile.enableLogging(true);       // Standardmäßig deaktiviert
```

Dabei werden während der Berechnung zwei Logdateien erzeugt. Die erste Datei heißt „Energy.log", in ihr können alle aufgetretenen Ereignisse nachgelesen werden. Darunter zählen Auffälligkeiten in der PDB-Datei, wie auch Probleme bei der Berechnung. Die zweite Datei heißt „Error.log" und wie der Name schon erahnen lässt werden in dieser Datei Fehler abgelegt, welche zum Abbruch der Berechnung führten. Eine solche Ursache könnte sein, dass unbekannte Aminosäuren, darunter fallen Aminosäuren, die nicht zu den 20 kanonischen Aminosäuren gehören, gefunden wurden (In diesen Fällen ist es derzeit noch nicht möglich ein Energieprofil zu erzeugen).

	x-Buchstaben-Code			x-Buchstaben-Code	
Aminosäure	**3**	**1**	**Aminosäure**	**3**	**1**
Alanin	Ala	A	Leucin	Leu	L
Arginin	Arg	R	Lysin	Lys	K
Asparagin	Asn	N	Methionin	Met	M
Asparaginsäure	Asp	D	Phenylalanin	Phe	F
Cystein	Cys	C	Prolin	Pro	P
Glutamin	Gln	Q	Serin	Ser	S
Glutaminsäure	Glu	E	Threonin	Thr	T
Glycin	Gly	G	Tryptophan	Trp	W
Histidin	His	H	Tyrosin	Tyr	Y
Isoleucin	Ile	I	Valin	Val	V

Die 20 kanonischen Aminosäuren mit Name, 3- sowie 1-Buchstaben-Code

Nun kann man beginnen ein Energieprofil zu erzeugen. Möglich ist das mittels drei verschiedener Funktionen, die sich allerdings nur in der Art des Aufrufs unterscheiden.

```
eProfile.generateProfile(strSource, strChain, bTm);  // (1)
eProfile.generateProfile(strSource, strChain);       // (2)
eProfile.generateProfile(strSource);                 // (3)
```

Der erste Parameter (`strSource`) ist ein String und zwingend notwendig. Er spezifiziert die Quelle, die zur Berechnung dient. Dabei ist es möglich eine lokale PDB-Datei zu wählen (z.B.: „D:\Daten\PDB\1a3h.pdb") oder auch eine URL als String anzugeben (z.B.: „http://www.pdb.org/pdb/files/1A3H.pdb").

Der zweite Parameter (`strChain`) ist ebenfalls ein String und identifiziert die Kette, die berechnet werden soll. Wenn unabhängig von der Kette das Energieprofil über das gesamte Protein erzeugt werden soll, kann als Wert ein leerer String oder ein Unterstrich („_") angegeben werden. Bei der Berechnung einer Kette werden räumlich angrenzende Aminosäuren anderer Ketten, die innerhalb der 8Å-Kugel liegen mit in die Berechnung einbezogen. Die Angabe einer speziellen Kette ist nützlich, wenn aus Zeitgründen nur ein Teil eines sehr umfangreichen Proteins berechnet werden soll und es für eine einmalige Benutzung ausgelegt ist. Wenn das Energieprofil zu späteren Zeitpunkten weiter verwendet werden soll, wird strengstens empfohlen die Angabe der Kette weg zu lassen und das gesamte Protein zu berechnen. Das betrachten einer einzelnen Kette eines Proteins ist auch zu einem späteren Zeitpunkt innerhalb der Energiedaten möglich. Auf diese Möglichkeit wird am Ende dieses Kapitels noch einmal näher eingegangen.

Der dritte Parameter (`bTM`) ist ein boolscher Wert und identifiziert die Quelle als ein transmembranes (`true`) oder ein nicht-transmembranes (`false`) Protein.

Die Funktion (3) ruft die Funktion (2) auf und lässt das gesamte Protein berechnen. Dabei wird vorher versucht die Art des Proteins zu bestimmen. Dazu wird die PDB-ID des Proteins in der PDB-TM („http://pdbtm.enzim.hu") nachgeschlagen. Falls sich ein Eintrag dazu finden lässt, wird vorausgesetzt, dass das Protein ein transmembranes Protein ist, andernfalls wird vermutet, dass es sich um ein nicht-transmembranes Protein handelt. Allerdings ist diese Methode nicht 100%ig zuverlässig. Ist die Art des Proteins bekannt, so empfiehlt es sich die Funktion (1) zu benutzen und mittels des dritten Parameters die entsprechende Art zu bestimmen.

Anschließend an die Berechnung kann man sich das Ergebnis auf drei verschiedene Weisen zugänglich machen. Die ersten beiden Varianten dienen einer Ausgabe im Textformat und der schnellen Ansicht des berechneten Energieprofils:

```
eProfile.writeCleanEnergyProfile(strDestination);     // (4)
eProfile.writeDetailedEnergyProfile(strDestination); // (5)
```

Die erste Variante (4) schreibt nur die nötigsten Informationen in die Zieldatei, welche über den Parameter „`strDestination`" festgelegt wird und erzeugt somit eine ep-Datei. Die zweite Variante (5) enthält den gleichen Informationsgehalt, wie auch (4), fügt jedoch noch einige zusätzliche Informationen an und erzeugt eine ep2-Datei.

Die letzte und umfangreichste Möglichkeit an das Energieprofil zu kommen, ist in Form der internen Energiedaten. Diese können mit folgender Funktion geholt werden:

```
EData eData = eProfile.getEnergyData();
```

Die Energiedaten können dann weiter verwendet werden und bieten sich an, wenn mit dem Energieprofil weiter gearbeitet werden soll. Man kann diese Energiedaten serialisiert ablegen (eps-Dateien) und natürlich auch wieder einlesen:

```
EData.save(eData.getPDBID() + ".eps", eData);
EData eDataLoaded = EData.load(eData.getPDBID() + ".eps");
```

Um mit den Energiedaten zu arbeiten, bieten diese eine Reihe von Funktionen an. Um die Funktionen (4) und (5) nachzuempfinden, sowie eine Ausgabe in Textform zu erhalten dienen die beiden toString-Funktionen, welche auch von (4) und (5) aufgerufen werden:

```
String strEP            = eData.toSimpleString();     // ep-Ausgabe
String strEP2           = eData.toDetailedString();   // ep2-Ausgabe
```

Die elementaren Aussagen zu einem Energieprofil können über folgende Möglichkeiten abgefragt werden:

```
String strID            = eData.getPDBID();           // (6)
String strSequence      = eData.getSequence();        // (7)
int nSize               = eData.getSize();            // (8)
boolean bTM             = eData.isTransmembrane();    // (9)
String strInfos         = eData.getInfos();           // (10)
```

(6) liefert die PDB-ID, welche dem Energieprofil zu Grunde liegt. (7) liefert die Sequenz und (8) die Länge der Sequenz bzw. die Länge des Energieprofils. Mit Hilfe von (9) kann überprüft werden, ob die Energiedaten von einem transmembranen oder nicht-transmembranen Protein stammen. Falls bei der Berechnung Ereignisse aufgetreten sind oder zusätzliche Informationen vorhanden waren, können diese über (10) erfragt werden.

Um an die eigentlichen Energiewerte zu gelangen kann man die folgende Funktion mit der Angabe des Index (von 0 bis `nSize - 1`) verwenden:

```
Double dEnergy          = eData.getEnergy(nIndex);    // (11)
```

Der Energiewert wird nicht mit dem primitiven Datentyp double dargestellt sondern mittels des Double-Objekts. Die Begründung dafür liegt in der Berechnung des Energieprofils. Vor allem bei transmembranen Proteinen ist es möglich, dass es unbekannte Bereiche gibt, welche nicht klar definiert werden können. An diesen Stellen kann kein Energiewert zugeordnet werden. Da diese Stellen jedoch trotzdem verzeichnet werden sollen, wird der Energiewert an den Stellen auf `null` gesetzt. Da auf Grund des normalen Energiespektrums auch durchaus der Wert 0.0 für die Energie auftreten kann, konnte dieser Wert nicht als „Nicht berechenbar" genutzt werden. Somit hat das Double-Objekt über den Wert `null` einen weiteren Zustand, der über den primitiven Datentyp nicht abgebildet werden kann.

Werden weitere Informationen für eine Stelle im Energieprofil benötigt, kann über folgende Funktion auch das Atom mittels des Index (von 0 bis `nSize - 1`) geholt werden. Das Atom steht dabei stellvertretend für die Aminosäure an dieser Stelle:

```
Atom aAtom              = eData.getAtom(nIndex);
```

Die Klasse „Atom" verfügt über eine Reihe von Attributen, die in dem Zusammenhang mit den Energiedaten allerdings keine direkte Beziehung eingehen. Im folgendem werden die wichtigsten Funktionen vorgestellt, die für die Energiedaten von Bedeutung sind.

```
String strRes           = aAtom.getResidueName();           // (12)
String strResOneLetter  = aAtom.getResidueNameOneLetter();  // (13)
int nResNo              = aAtom.getResidueSequenceNumber(); // (14)
String strChain         = aAtom.getChainIdentifier();       // (15)
String strSecStructure  = aAtom.getSecStructure();          // (16)
Double dEnergy          = aAtom.getEnergy();                // (17)
```

Diese Funktionen liefern Informationen über die Aminosäure an sich, also den Name im 3-Buchstaben-Code (12), im 1-Buchstaben-Code (13), sowie die Nummer (14). Über (15) kann die zugehörige Kette ermittelt werden. (16) liefert die Sekundärstruktur, zu der die Aminosäure gehört und (17) liefert den Energiewert für diese Stelle bzw. Aminosäure. Die Funktion (17) wird intern von der Funktion (11) aufgerufen um an den Energiewert zu gelangen.

Relativ am Anfang des Kapitels wurde beschrieben, wie man einzelne Ketten eines Proteins berechnen lassen kann und dass diese Variante nur in speziellen Fällen angewendet werden sollte. Die folgende Funktion bietet die bessere Alternative. Mit dieser Funktion kann aus berechneten Energiedaten eine einzelne Kette (`strChain`) extrahiert werden:

```
EData eDataChain  = eData.getChain(strChain);
```

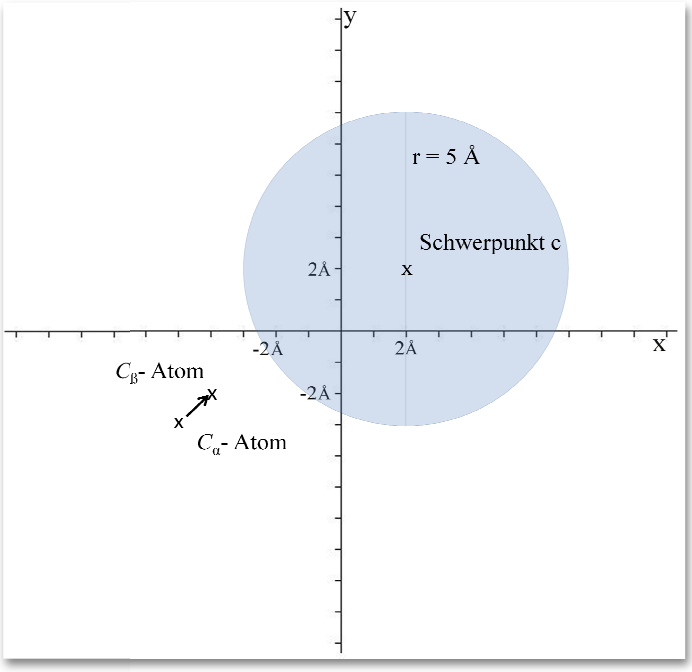
4.1.2. Funktionsweise

Wie im vorherigen Kapitel schon erwähnt, basiert eProfile auf dem von Dr. Frank Dressel entwickelten Algorithmus, wurde jedoch modifiziert. Im folgendem soll die grundlegende Funktionsweise erläutert werden.

Zur Beschreibung der freien Energie einer Aminosäure müssen inter- sowie intramolekulare Wechselwirkungen betrachtet werden. Die intermolekularen Interaktionen werden durch Wechselwirkungen mit dem Lösungsmittel beschrieben. Der Ansatz ist eine Statistik zur Bewertung der Aminosäuren. Diese beruht auf dem Ansatz der Zuweisung einer Innen- oder Außenposition, einer betrachteten Aminosäure i in Bezug auf das Protein. Dabei werden das C_α-Atom sowie das C_β-Atom dieser Aminosäure i betrachtet. Um sie wird eine Kugel mit dem Radius von 10 Å (Ångström) gelegt und der Schwerpunkt aller C_α-Atome der Aminosäuren in dieser Kugel berechnet. Ångström ist eine Längeneinheit und entspricht etwa dem Durchmesser eines Atoms (1 Å = 0,1 nm). Um die Aminosäure als Innen anzusehen, muss einer der beiden Terme erfüllt sein:

$$|C_\alpha - c| < 5\ \text{Å} \ \vee \ (C_\alpha - C_\beta)(C_\alpha - c) < 0\ \text{Å}$$

Die Variable „c" beschreibt hierbei den Schwerpunkt. Der erste Term verarbeitet den Abstand des C_α-Atoms der Aminosäure i zum Schwerpunkt. Wenn dieser kleiner als 5 Å ist, wird i als Innen gewertet. Der zweite Term beschreibt die räumliche Orientierung der Aminosäure i bzw. ihrer Seitenkette unter Einbeziehung des C_β-Atoms. Dies ist in folgender Abbildung verdeutlicht:



Darstellung der räumlichen Orientierung und Verarbeitung des Terms: $(C_\alpha - C_\beta)(C_\alpha - c) < 0$ Å

Der Algorithmus verarbeitet nun die intramolekularen Wechselwirkungen, also die Wechselwirkungen zwischen den Aminosäuren. Dabei durchläuft er jede Aminosäure in der Sequenz, legt eine Kugel von 8 Å um sie herum und betrachtet dabei die innerhalb dieser Kugel auftretenden Wechselwirkungen zwischen der aktuellen und allen benachbarten Aminosäuren.

Der implementierte Algorithmus greift über zwei verschiedene Dateien auf die Innen/Außen-Statistik zu und erstellt eine zugehörige Lookup-Tabelle mit Energiewerten für nicht-

transmembrane (aus der inoutGlob.txt), sowie zwei Lookup-Tabellen für transmembrane Proteine (aus der inoutMem.txt). Für transmembrane Proteine müssen zwei Tabellen angelegt werden, da man innerhalb eines transmembranen Proteins in zwei verschiedene Bereiche unterscheiden kann. Das ist einmal der innere und einmal der äußere Bereich. Es gibt allerdings auch undefinierte bzw. unbekannte Bereiche, welche im Energieprofil dann als Lücken gekennzeichnet sind. Die Initialisierung der Lookup-Tabellen, sowie der restlichen globalen Attribute wird unmittelbar im Konstruktor (`new EProfile()`) vorgenommen.

Startet man die Erstellung eines Energieprofils, werden zunächst alle Listen und Werte die zur Erstellung benötigt werden zurückgesetzt. Damit ist die Möglichkeit folgender Batch-Bearbeitung gegeben:

```
EProfile eProfile = new EProfile();

for (...)
{
      eProfile.generateProfile("<Pfad zur PDB-Datei>", "_", false);

      EData eData = eProfile.getEnergyData();
      EData.save(eData.getPDBID() + ".eps", eData);
}
```

Anschließend wird die übergebene PDB-Datei geparst und alle interessanten Informationen werden in einem `PDBFile`-Objekt abgelegt. Wenn es sich um ein transmembranes Protein handelt, wird auf gleiche Weise wie bei der PDB-Datei ein `PDBTMFile`-Objekt erzeugt.

```
// Ausschnitt aus der Funktion parsePDBFile(String strSource)

PDBParser parserPDB = new PDBParser();
parserPDB.setFile(strSource);
parserPDB.parsePDB();

filePDB = parserPDB.getPDBFile();
```

Das `PDBTMFile`-Objekt wird durch direktes parsen von der Seite der PDBTM (pdbtm.enzim.hu) geladen. Falls es sich um kein transmembranes Protein handelt wird dieses `PDBTMFile`-Objekt `null` gesetzt.

```
if (tm)
{ parsePDBTM(); }
else
{ filePDBTM = null; }
```

An dieser Stelle sind alle Grundlagen für das Erstellen eines Energieprofils vorbereitet und das eigentliche Generieren beginnt. Dazu werden als erstes alle Cα-Atome aus dem `PDBFile`-Objekt geholt und in einem Array abgespeichert. Die Cα-Atome stehen dabei stellvertretend für die Aminosäure an dieser Position. Dabei werden noch verschiedene Bereinigungen des Datensatzes vorgenommen. Mehrere Modelle (Dopplungen) und auch das Auftreten alternativer Positionen einzelner Atome wird ausgeschlossen. Diese Daten bilden die Grundlage für die Berechnung.

In einer geschachtelten Schleife werden die Atomdaten verarbeitet. In der ersten Schleife werden die Positionen der Cα-Atome von der betrachteten Aminosäure i, der nachfolgenden sowie der vorangegangenen Aminosäure genutzt. Dabei wird ein Abstand von 8 Å zu den C_α-

Atomen überprüft und wenn dieses Kriterium erfüllt ist, wird zur aktuellen Aminosäure das C_β-Atom anhand der beiden benachbarten Aminosäuren berechnet.

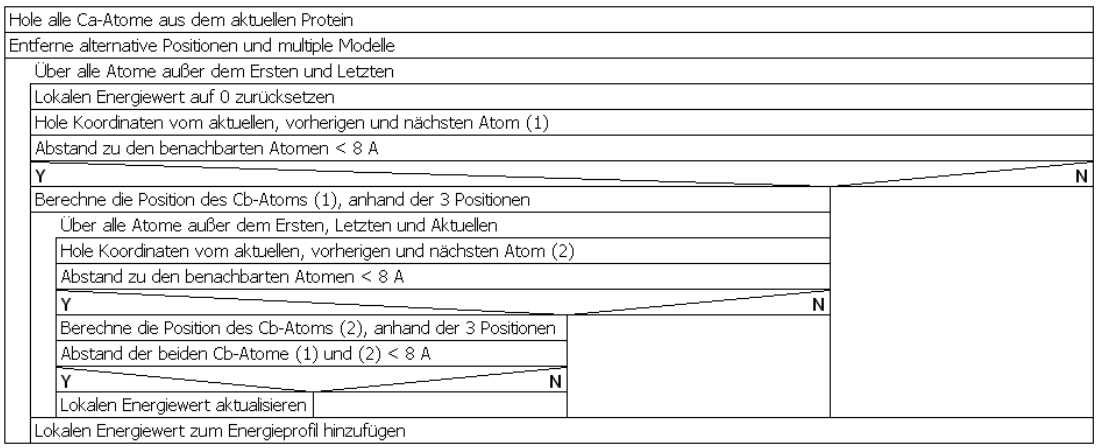
In der zweiten Schleife werden erneut die Positionen der C_α-Atome durchlaufen (die aktuelle Aminosäure wird ausgeschlossen). Dabei werden wiederum die Positionen der C_α-Atome der aktuellen Aminosäure j, sowie seinem Vorgänger und Nachfolger in einem Abstand von 8 Å bestimmt. Wenn diese Voraussetzung erfüllt ist, wird daraufhin das C_β-Atom berechnet und mit dem C_β-Atom aus der ersten Schleife verglichen. Wenn der Abstand kleiner oder gleich 8 Å ist liegt die Aminosäure j innerhalb der zu betrachtenden Kugel.

An dieser Stelle kommen die Lookup-Tabellen für die Energiewerte zum Einsatz. Falls es sich um ein nicht-transmembranes Protein handelt, wird je der Energiewert der aktuellen und der interagierenden Aminosäure nachgeschlagen und zum lokalen Energiewert addiert.

Handelt es sich um ein transmembranes Protein, so wird anhand des `PDBTMFile`-Objekts die Position der jeweilig aktuellen und der interagierenden Aminosäure herausgesucht und der Energiewert aus der entsprechenden Lookup-Tabelle entnommen und zum lokalen Energiewert addiert.

Bevor sich die äußere Schleife der nächsten Aminosäure widmet, wird der lokale Energiewert als Energiewert für die Aminosäure in die Energiedaten aufgenommen. Wenn die äußere Schleife durchlaufen wurde, stehen somit für alle Aminosäuren Energiewerte fest. Die Energiedaten bilden somit eine Abfolge von Energiewerten, das Energieprofil.

Wie man an diesem Algorithmus erkennen kann existiert für die erste Aminosäure kein Vorgänger, sowie für die letzte Aminosäure kein Nachfolger. Aus diesem Grund werden bei der Berechnung diese beiden Stellen abgeschnitten und können im Energieprofil nicht mit berechnet werden.



Struktogramm zeigt den Grobablauf der Berechnung des Energieprofils (Ca = C_α, Cb = C_β, A = Å)

4.2. eVis

4.2.1. Nutzung

Das Visualisierungstool „eVis" (energy visualisation) ist ein Applet und muss somit in den HTML-Code eingebunden werden, wenn man es nutzen möchte. Das folgende Beispiel zeigt eine Möglichkeit, wie sie zahlreich auf dem Server verwendet wurde:

```
<applet name="EVis" archive="eVis.jar, eData.jar, Helper.jar"
codebase="/Epros/resources/jars"
code="de.hsmw.bioinformatics.evis.gui.EVisApplet.class" width="100%"
height="310">

<!-- Parameterbereich -->

</applet>
```

Das Namens-Attribut (`name="EVis"`) ist notwendig um die korrekte Interaktion mit Jmol zu gewährleisten. Sollte „eVis" alleine stehen, so kann dieses Attribut entfallen.

Die Angabe der zu ladenden Archive ist zwingend und muss die drei angegebenen Archive umfassen (`archive="eVis.jar, eData.jar, Helper.jar"`) andernfalls kann das Applet nicht laufen. `eVis.jar` stellt dabei das Applet an sich dar, wohingegen `EData.jar` und `Helper.jar` Hilfsklassen enthalten, die von „eVis" dringend benötigt werden.

Die Positionsangabe des Ordners (`codebase="/Epros/resources/jars"`), in dem die Archive zu finden sind kann mitunter entfallen, es wird aber empfohlen sie mit anzugeben.

Im Code-Attribut ist die Startklasse zu finden, welche beim Initialisieren des Applets zu laden ist (`code="de.hsmw.bioinformatics.evis.gui.EVisApplet.class"`). Diese muss mit vollständigem Paketname angegeben werden, falls sie nicht in erster Ebene im Archiv liegt.

Die letzten beiden Attribute spezifizieren die Breite (`width="100%"`) und die Höhe (`height="310"`) des Applets in Bezug auf die umliegenden Elemente. In diesem Fall entspricht die Höhe von 310 der Angabe für die Anzeige eines einzelnen Datensatzes. Sollen zwei Datensätze angezeigt werden empfiehlt sich eine Höhe von 370.

Eingeschlossen in die Applet-Tags (`<applet …></applet>`) ist der Parameterbereich zu finden. Ohne diesen ist „eVis" nicht in der Lage Daten anzuzeigen. In diesen Bereich können beliebig viele Einträge der folgenden Form enthalten sein:

```
<param name="Name" value="Value" />
```

Wobei `Name` und `Value` durch entsprechende Informationen ersetzt werden müssen. Die folgende Tabelle zeigt alle möglichen Kombinationen, für erlaubte Parameterübergaben an „eVis".

Name	Beschreibung (# kann für die Zahlen 0 oder 1 stehen)
eps#	Pfad und ID zu einer *.eps-Datei als Datensatz #
chain#	Anzuzeigende Kette für Datensatz #
suffix#	Jmol-Suffix für den Datensatz #
manually#	String in einem speziellen Format als Datensatz #
align	Alignment für 2 Datensätze
mutation_mode	Aktivierung eines speziellen Darstellungsmodus für Mutationen

Dabei ist zu beachten, dass nicht alle Parameter beliebig kombiniert werden können. Spezielle Einsatzgebiete können im Kapitel 6 – JSP nachgeschlagen werden. Ob man „eps#" oder „manually#" nimmt um den Datensatz zu übergeben ist egal, jedoch wird „eps#" vor „manually#" bevorzugt, falls man beide angibt. Die eps#-Parameter müssen in Form von URL-Adressen übergeben werden, die auf eine EPS-Datei verweisen.

```
<!-- Gültig -->
<param name="eps0" value="Pfad/1DSL" />
<param name="eps1" value="Pfad/1AMM" />

<!-- Gültig -->
<param name="manually0" value="ETAAAKFERQ###0 2 7 5.7 3 2 0 0 0 4" />
<param name="manually1" value="PYVPVHFDAS###3 10 3 6 7 4 11 2 0 1" />

<!-- manually0 wird nicht angezeigt, da beide die selbe ID haben -->
<param name="eps0" value="Pfad/1DSL" />
<param name="manually0" value="ETAAAKFERQ###0 2 7 5 3 2 0 0 0 4" />
```

Der Syntax für einen „manually#"-String ist folgendermaßen beschrieben.

```
Getrennt von Rauten (#):

Sequenz
Sekundärstruktur (optional)
Aminosäurennummern (optional, mit Leerzeichen getrennt)
Energiewerte (mit Leerzeichen getrennt)
```

Die Sequenz ist dabei eine Abfolge von Aminosäuren im 1-Buchstaben-Code, danach folgt als Trennzeichen eine Raute (#). Die Sekundärstruktur kann aus einer Kombination von [Hh] für Helix, [SsFfEe] für Sheet und [CcLl-] für Coil bestehen oder auch ganz entfallen (in diesem Fall werden Fragezeichen als Sekundärstruktur angezeigt). Anschließend folgt als Trennzeichen wieder eine Raute und die Aminosäuren-Nummern, die durch Leerzeichen getrennt werden. Als letztes folgt daran wieder eine Raute und die Energiewerte, die ebenfalls durch Leerzeichen getrennt werden.

```
Beispiel:

AGV###-2.0 -3.2 -1.12
AGV#HHE##-2.0 -3.2 -1.12
AGV#HHE#1 2 3#-2.0 -3.2 -1.12
```

Die Angabe der Kette (chain#) macht nur für Datensätze Sinn, welche mit „eps#" beschrieben wurden und kann auch entfallen, wenn keine spezielle Kette hervorgehoben werden soll.

Die Angabe eines Suffix (suffix#) ist essenziel für die Kommunikation mit Jmol. Er muss dem Suffix des Jmol-Applets entsprechen.

Wenn zwei Datensätze nicht nur übereinander gelegt werden sollen, sondern auch gleich ein Alignment durchgeführt werden soll kann der Parameter „align" verwendet werden. Dieser besteht aus zwei alignierten Sequenzen, getrennt durch eine Raute (#).

```
Beispiel:

-ETAAAKFERQHMDSSTSAASS#DNSRYTHFLTQAYDAKPQG-RD
```

Für die Darstellung von Mutationen bzw. von Energiedifferenzen bietet es sich an, den Parameter „mutation_mode" zu nutzen (`value="true"`). Dieser nutzt eine spezielle Färbung und eine eigene Energieskala (von -10 bis 25).

Die Energieskala wird automatisch an die vorliegenden Daten angepasst. Handelt es sich um nicht-transmembrane Daten, so geht sie von -50 bis 10. Liegen transmembrane Daten vor, so geht sie von -15 bis 75. Bestehen die Daten aus je einer von beiden Arten, so geht die Skala von -60 bis 80. Es wurde sich gegen eine dynamische Skala vom kleinsten zum größten Wert entschieden, da dies ein falsches Bild von den Werten vermitteln würde, wenn man sich mehrere Energieprofile hintereinander oder nebeneinander anschaut.

4.2.2. Funktionsweise

Wie alle Applets in Java, so startet auch „eVis" nicht mit einem Konstruktor, sondern mit einer init-Methode, die vom Browser aufgerufen wird, sobald das Applet angefordert wird. Diese liest alle möglichen Parameter ein, die an das Applet übergeben werden können und erstellt daraus die Daten, die benötigt werden.

Als erste Besonderheit wird ein boolscher Wert auf true gesetzt, wenn die Übergabe von Suffixen stattgefunden hat, damit eine Kommunikation mit Jmol ermöglicht werden kann. Dies ist später wichtig für die Callback-Funktionen.

```
bJmolEnabled = (strSuffix0 != null || strSuffix1 != null);
```

Weiterhin werden in dieser init-Methode die Datensätze eingelesen. Wenn eps-Parameter übergeben wurden, dann werden die entsprechenden EPS-Dateien in ein „EData"-Objekt geladen und falls eine Kette angegebenen wurde, diese herausgelesen. Falls keine eps-Parameter übergeben wurden, wird auf manually-Parameter geprüft. Dabei wird ein „EData"-Objekt anhand des Parameters aufgebaut.

Die so initialisierten Daten werden an ein Panel (`EVisPanel`) weitergereicht und dieses wird anschließend als Oberfläche auf das Applet gesetzt.

Jedes mal wenn das Applet sichtbar wird (wenn das Fenster in dem das Applet liegt den Fokus erhält), wird die start-Methode des Applets aufgerufen. Im Fall von „eVis" übernimmt diese beim ersten Aufruf die Initialisierung der grafischen Oberfläche des Panels. Anschließend wird ein boolscher Wert gesetzt, damit das Panel nicht mehr initialisiert werden muss. Der Grund für die Initialisierung an dieser Stelle und nicht schon in der init-Methode liegt im prinzipiellen Informationsgehalt dieser beiden Methoden. Beim Aufruf der start-Methode ist die Breite und Höhe des Applets bekannt, in der init-Methode noch nicht. Dazu müssten die Angaben für die Größe mit in Form von Parametern übergeben werden.

Das Panel `EVisPanel` dient als Oberfläche für das Applet. Im Konstruktor werden ihm die Datensätze übergeben. Anhand deren wird die Energieskala eingestellt. Anschließend werden die Datensätze in verschiedene Container gelegt, die noch zusätzliche Informationen über den aktuell markierten Bereich, das genutzte Alignment, die Position fürs Scrollen und noch ein paar andere Werte enthalten. Der größte Teil der Oberfläche besteht aus einem Diagramm, dieses wird erzeugt indem die `paintComponent`-Methode der Klasse „JPanel" überschrieben wird. Die restlichen Komponente, genauer gesagt, die Steuerelemente (Scrollbar und Radiobuttons) im unteren Teil des Panels werden durch die start-Methode vom Applet aus initialisiert. Je nach

Einstellungen (z.B.: Aktivierung des „mutation_mode“) können dabei verschiedene Steuerelemente ausgeblendet werden.

Den wohl wichtigsten Teil von „eVis“ bildet das Diagramm, welches die Energiedaten visualisiert. Wie schon weiter oben beschrieben, geschieht dies durch das überschreiben der `paintComponent`-Methode. Diese wird immer aufgerufen, wenn das Applet sichtbar wird oder sich seine Größe ändert. Zusätzlich kann man über diese Methode den Aufruf von `repaint()` anstoßen. Dies geschieht in „eVis“ beispielsweise beim Scrollen des Sichtbereichs oder auch beim Markieren von Bereichen. Offensichtlich ist dies recht häufig der Fall. Aus diesem Grund wird eine Doppelpufferung verwendet. Das bedeutet, dass der eigentlich darzustellende Teil zuerst auf einer zweiten Zeichenebene in Form eines Hintergrundbildes gezeichnet wird. Sobald alle Zeichenoperationen abgeschlossen sind, wird dieses Hintergrundbild auf die eigentliche Zeichenebene übertragen. Dadurch werden unschöne Flacker-Effekte vermieden.

Beim Neuzeichnen der Oberfläche werden folgende vier Schritte einzeln ausgeführt:

1. Die Größe und die Position der Steuerelemente werden aktualisiert. Dieser Schritt ist elementar, falls sich die Größe des Applets ändert (zum Beispiel durch ändern der Größe des Browserfensters).

2. Der derzeit markierte Bereich wird für jeden Datensatz an der entsprechenden Position als grau unterlegte Fläche in das Diagramm eingezeichnet.

3. Das Koordinatensystem, die Energieskala und die Beschriftung der einzelnen Datenbereiche unter dem Diagramm werden eingezeichnet.

4. Die Energieprofile werden gezeichnet und die entsprechenden Datenbereiche werden gefüllt. In diesem vierten Schritt werden auch die Farben des derzeitig ausgewählten Farbschemas einbezogen um den entsprechenden Informationsgehalt darzustellen.

Damit eine Interaktion zwischen den Steuerelementen und der Oberfläche möglich ist werden verschiedene Listener benötigt. Darunter fallen für die Scrollbar ein „AdjustmentListener“ und für die RadioButtons ein „ActionListener“. Diese ermöglichen es bei einer Veränderung des Status verschiedene Methoden aufzurufen. So werden bei der Benutzung der Scrollbar die Anfangswerte der einzelnen Datencontainer neu gesetzt und anschließend wird über den Aufruf von `repaint()` eine Aktualisierung der Oberfläche angefordert. Nach dem gleichen Prinzip werden die Farbschemen genutzt. Sobald ein neues Farbschema ausgewählt wird, erhält die Variable für das aktuell benutze Farbschema (`nColorMode`) einen neuen Wert und es wird mittels `repaint()` eine Aktualisierung der Oberfläche angefordert.

Um das Markieren von Bereichen zu ermöglichen ist es notwendig einen weiteren Listener, den „MouseListener“ auf der Oberfläche anzumelden. Dies geschieht nur, wenn die Kommunikation mit Jmol aktiviert wurden ist. Dieser ermöglicht es Klicks mit der Maus zu untersuchen. Dabei werden in „eVis“ genau drei Maus-Ereignisse überwacht. Zum einen das Klicken und gedrückt halten der linken Maustaste. Weiterhin das Loslassen der linken Maustaste und das Klicken und Loslassen einer anderen Maustaste.

Die ersten beiden Ereignisse ermöglichen es im Zusammenspiel Bereiche zu markieren. Dabei wird anhand der Position der Maus über dem Diagramm bzw. dem Datenbereich der gewünschte Bereich bestimmt. Unterschiedlich wird dabei behandelt, ob nur ein Datensatz dargestellt wird, dann kann man im Diagramm selber und im Datenbereich markieren, oder ob

zwei Datensätze vorliegen, dann ist es nur im Datenbereich möglich Markierungen vorzunehmen. Die Position der Maus wird anschließend anhand der Anfangswerte aus den Datencontainern und der Breite der einzelnen Datenpunkte in eine absolute Position innerhalb des Datensatz umgerechnet. Wenn die Maustaste auch in einem gültigen Bereich wieder los gelassen wird, wird aus der berechneten Anfangsposition und der Endposition der innerhalb liegende Bereich markiert und es wird wieder mittels `repaint()` eine Aktualisierung der Oberfläche angefordert. Die Markierung selber wird in den entsprechenden Datencontainern abgelegt. Innerhalb der Datencontainer werden die Markierungen zusätzlich zur absoluten Position mit der entsprechenden Aminosäuren-Nummer und Kettenzugehörigkeit in einem eigenen Container verwaltet.

Das dritte Mausereignis bereinigt die Markierungscontainer und setzt sie auf ihren Ausgangsstand zurück. Damit ist es also möglich alle derzeitigen Markierungen zu löschen. Damit die Änderungen sichtbar werden, wird auch hier wieder mittels `repaint()` eine Aktualisierung der Oberfläche angefordert.

4.2.3. JavaScript, Jmol und Callback-Funktionen

Schon mehrmals wurde die Möglichkeit der Interaktion mit Jmol erwähnt. Dies ist mittels Callback-Methoden von Jmol und über eine JavaScript-Schnittstelle möglich.

Um diesen Vorgang besser zu verstehen wird kurz eine Einführung in die Möglichkeiten des Jmol-Applets getätigt. Um Jmol in eine HTML-Seite einzubinden bietet es eine eigene Schnittstelle, basierend auf JavaScript an. Damit ist es möglich über einfache Script-Befehle das Applet zu steuern. Auf den Seiten von „eProS" sieht diese Initialisierung des Jmol-Applets ähnlich dem folgendem Template aus:

```
<script type="text/javascript">
      var file = "Pfad/1DSL.pdb";                              // (1)
      jmolInitialize("/Epros/resources/jars/jmol");            // (2)
      jmolApplet(400, "set showFrank false; set disablePopupMenu true;
set PickCallback \"atomPicked\"; load \"" + file + "\"; set picking
measure distance; cartoon only; color structure; javascript
jmolLoaded('uniqueSuffix', 'chain');", "uniqueSuffix");    // (3)
</script>
```

Über die file-Variable (1) wird der Pfad zu einer PDB-Datei gesetzt. Wie genau dieser Pfad gebildet wird, kann im Kapitel 6.2. – Die Tool-Seiten nachgeschlagen werden. Die erste Anweisung für die Nutzung jedes Jmol-Applets ist (2). Dabei wird als Parameter der Pfad angegeben, in dem das Jmol-Applet liegt, genauer gesagt das JAR-Archiv des Applets. Mit der Anweisung (3) wird das Jmol-Applet an dieser Stelle eingebunden. Der erste Parameter stellt dabei die Größe des Applets dar. Der zweite Parameter ist ein auszuführendes Jmol-Script, welches während des Initialisierens ablaufen soll. Der dritte Parameter ist ein eindeutiger Bezeichner, mit dem man das Applet im Folgenden ansprechen kann (entspricht dem name-Attribut im Applet-Tag).

Das Initialisierungsscript (Parameter 2 aus (2)) sorgt für das Aussehen von Jmol, wie es auf „eProS" zu finden ist:

```
set showFrank false;                // Blendet das Jmol-Logo aus
set disablePopupMenu true;          // Deaktiviert das Kontextmenü
```

```
set PickCallback \"atomPicked\";    // (4)
load \"" + file + "\";              // Lädt die Datei in Jmol
set picking measure distance;       // Blendet Abstandsmessungen aus
cartoon only;                       // Aktiviert die Cartoondarstellung
color structure;                    // Färbt nach Sekundärstrukturen
javascript jmolLoaded('uniqueSuffix', 'chain');      // (5)
```

(4) registriert eine Callback-Funktion mit dem Name „atomPicked" für alle Auswahlvorgänge in Jmol. Dies bedeutet, dass wenn in Jmol ein Atom bzw. eine Aminosäure durch die Cartoon-Darstellung angeklickt wird, wird die Funktion „atomPicked" aufgerufen, der verschiedene Parameter übergeben werden. Damit ist es also möglich eine Kommunikation von Jmol zu einem JavaScript einzurichten.

Die Anweisung (5) besagt, dass nach allen Initialisierungen ein JavaScript mit dem Namen „jmolLoaded" ausgeführt werden soll und es die Parameter „uniqueSuffix" und „chain" übergeben bekommt.

Diese beiden Funktionen sollen im folgendem näher betrachtet werden.

```
function atomPicked(applet, infos, number) {
```

Dem Script werden genau drei Parameter übergeben. Der erste Parameter (`applet`) liefert den Namen (den Suffix) des Applets, damit ist eine Zuordnung der Herkunft möglich. Der zweite Parameter (`infos`) liefert Informationen über die ausgewählte Aminosäure. Der dritte Parameter (`number`) liefert die Atom-ID, die für die weiteren Vorhaben nicht interessant ist.

```
infos             = infos.substring(infos.indexOf("]") + 1);
var nResNo        = infos.substring(0, infos.indexOf(":"));
var strChain      = infos.substring(infos.indexOf(":") + 1,
                    infos.indexOf(":") + 2);

if (strChain == ".") { strChain = " "; }
if (strChain == " ") { strChain = ""; }
```

Aus den Informationen zur aktuellen Aminosäure wird die Aminosäuren-Nummer (`nResNo`) und die zugehörige Kette (`strChain`) ausgelesen. Falls keine Kettenangabe vorhanden ist, wird die Kette weggelassen und der komplette Datensatz durchsucht. Weitere Informationen werden nicht benötigt.

```
var strIndex      = applet.substring(applet.length - 1);
```

Um auch mehrere Jmol-Applets mit „eVis" die Kommunikation zu ermöglichen erhalten alle Jmol-Applets einen Index an den Name angehangen, der dem jeweiligen Datensatz in „eVis" passt. Somit können Auswahlen an den richtigen Datensatz weiter gegeben werden.

```
document.applets["EVis"].selectRes(strIndex, strChain, nResNo); }
```

Diese letzte Angabe spricht „eVis" an. Die Methode „selectRes" des „eVis"-Applets wird mit den drei Parametern aufgerufen. Dabei ist die Voraussetzung, dass das name-Attribut im Applet-Tag von „eVis" den Wert „EVis" besitzt. Dieser Schritt schließt den ersten Kreis und ermöglicht so die Kommunikation von Jmol über JavaScript zu „eVis".

Das zweite Script (5), was Jmol ausführen soll, sobald es fertig initialisiert ist sieht folgendermaßen aus:

```
<!-- Globale Variablen -->
loadedJmol = new Array();
loadedJmol[0] = new Object();
loadedEvis = "";
```

Diese globalen JavaScript-Variablen ermöglichen es den Lade-Status der Applets zu überwachen.

```
function jmolLoaded(suffix, chain) {
loadedJmol[0][suffix] = "loaded";
```

Sobald das Jmol-Applet fertig initialisiert wurde markiert es sich selber als fertig geladen.

```
if (chain != "")
{ executeJmolScript("display :" + chain, suffix); }        // (6)
```

Falls eine Kette angegeben ist, wird über die Funktion „executeJmolScript" ein Jmol-Script ausgeführt. In diesem Fall bewirkt es, dass nur die aktuelle Kette angezeigt wird und alle anderen verborgen werden.

```
if (loadedEvis == "loaded")
{ document.applets["EVis"].callBackColor(); }}
```

Die letzte Anweisung in der Funktion prüft, ob das auf dieser Seite stehende „eVis"-Applet bereits fertig geladen wurde. Falls das der Fall ist, wird die Methode „callBackColor" aufgerufen. Damit sich das „eVis"-Applet als geladen melden kann, wird innerhalb der start-Methode die folgende JavaScript-Funktion aufgerufen:

```
function evisLoaded(suffix) {
loadedEvis = "loaded";

if (loadedJmol[0][suffix] == "loaded")
{ document.applets["EVis"].callBackColor(); }}
```

Es setzt damit den eigenen Status auf „fertig" und prüft den Status des Jmol-Applets. Falls es auch schon „fertig" ist, wird ebenfalls die Methode „callBackColor" aufgerufen. Mit dieser Gegenprüfung ist in den meisten Fällen gewährleistet, dass die Methode „callBackColor" aufgerufen wird, sobald beide Applets bereit sind.

Die „callBackColor"-Methode aus dem „eVis"-Applet holt sich zuerst das aktuell gewählte Farbschema aus „eVis". Anhand dieses Farbschemas wird ein Jmol-Script zusammengebaut. Je nach Auswahl des Schemas (nur zutreffend auf Quantil- und auf Differenz-Färbung) werden die Energiewerte aus den Energiedaten geholt und entsprechend der Färbung verarbeitet. Das aufgebaute Jmol-Script wird an alle registrierten Jmol-Applets, über die Angabe des Suffix, mittels der JavaScript-Schnittstelle geschickt.

```
getAppletContext().showDocument(new
URL("javascript:executeJmolScript(\"" + script + "\", \"" + suffix +
"\")"));
```

Die Funktion „executeJmolScript" (auch in (6) benutzt) ruft nur die von Jmol bereit gestellte Funktion „jmolScript(script, suffix)" auf.

Nachdem das Farbschema von „eVis" zu Jmol übertragen ist, wird ein zweites Jmol-Script zusammengebaut um die Markierung aus „eVis" an Jmol zu schicken. Dabei werden die Markierungsdaten aus den jeweiligen Datencontainern geholt und entsprechend umgewandelt,

damit alle markierten Aminosäuren in Jmol ausgewählt werden können. Die Auswahl wird danach umgekehrt. Das bedeutet, dass nun alle nicht markierten Aminosäuren ausgewählt sind und anschließend halbtransparent dargestellt werden. Damit ist der zweite Kreis geschlossen und „eVis" kommuniziert nun mittels der JavaScript-Schnittstelle mit Jmol.

Sowohl der „ActionListener", der „MouseListener" und die Methode „selectRes" in „eVis" rufen ebenfalls diese „callBackColor"-Methode auf, sobald eine Änderung eingetreten ist.

4.3. eMut

Das Tool „eMut" (energy mutation) dient dem Auffinden und Analysieren von Mutationen in zwei Proteinen. Die aktuelle Version unterstützt dabei ausschließlich Punktmutationen von zwei gleichlangen Proteinen (bzw. Sequenzen), andere Mutationen können nicht bearbeitet werden.

4.3.1. Nutzung

Um „eMut" zu Benutzen sind folgende Schritte notwendig:

```
// Instanziieren der Klasse
EMut eMut = new EMut();
```

Im Konstruktor werden keine Initialisierungen vorgenommen, damit dient es nur der Instanziierung der Klasse.

```
eMut.setData(eData[0], eData[1]);
```

Diese Methode erwartet zwei Energiedaten als Parameter, welche von den punktmutierten Proteinen stammen. Da in dieser Methode alle Initialisierungen vorgenommen werden ist die Möglichkeit folgender Batch-Bearbeitung gegeben:

```
EMut eMut = new EMut();

for (...)
{
      eMut.setData(eData[0], eData[1]);
      // Daten verarbeiten
}
```

Damit sind alle notwendigen Vorbereitungen getroffen und alle Variablen und Strukturen initialisiert. „eMut" bietet zwei grundlegende Varianten an, mit denen die Mutationsinformationen aufbereitet werden können. Zum Einen sind das die generellen Informationen über die beiden Proteine, zum Anderen sind das Detailinformationen über die Mutationsstellen. Die generellen Informationen können folgendermaßen abgerufen werden:

```
String[] strMutationOutput = eMut.getSequenceMutation();
```

Das gelieferte Array besteht aus fünf Einträgen. Die ersten zwei Zellen liefern die Sequenzen der Proteine. Die dritte Zelle beinhaltet eine Mutationszeile. Gekennzeichnet ist diese durch eine Abfolge von Bindestrichen (-) für übereinstimmende Aminosäuren in den Sequenzen, sowie unterbrechende Rauten (#) für Mutationsstellen. Der vierte Eintrag im Array stellt die gemeinsame Sekundärstruktur dar. Darunter ist zu verstehen, dass für alle übereinstimmenden Sekundärstrukturen diese auch in dem String zu finden sind und abweichende Strukturen mit einem Fragezeichen (?) gekennzeichnet werden. Der letzte Eintrag enthält, mit Leerzeichen getrennt, die Energiedifferenzen der beiden Proteine.

```
// Beispiel eines fiktiven Arrays
strMutationOutput[0] // ETAAAKFERQHMDSSTSAASS
strMutationOutput[1] // ETALAKFERQHMDSATSAASS
strMutationOutput[2] // ---#----------#------
strMutationOutput[3] // HHH??ccEEEEccccc???HH
strMutationOutput[4] // 0 0 1 5 3 1.2 3 0 0 0 2 1 3 5 4 0 0 1 2 3 1.72
```

Mit diesen Ausgaben können die Mutationsstellen schon grob umrissen werden. Um detailierte Informationen über die einzelnen Mutationsstellen (Punktmutationen) zu erhalten, können folgende Methoden verwendet werden.

```
int nMutations    = eMut.getMutations();
Mutation m        = eMut.getMutation(nIndex);
```

Die absolute Anzahl an Mutationsstellen (`nMutations`) kann benutzt werden, um die einzelnen Mutationen anhand des Index zu erhalten. Dabei beginnt der Index bei 0 und endet bei (`nMutations` - 1). Es ist also auch durchaus möglich alle Mutationen in einer Schleife zu durchlaufen.

```
Mutation m = null;

for (int i = 0; i < nMutations; i++)
{
      m = eMut.getMutation(i);
      // Daten verarbeiten
}
```

Die Informationen die eine Mutation enthält sind zum Einen die Aminosäure, in welcher die Mutation erfolgt ist:

```
Atom aMutation = m.getMutation();
```

Zum Anderen ist es möglich, eine detailierte Liste auszugeben, in welcher für alle Energiedifferenzen der Abstand zur Mutationsstelle (in Å) berechnet wurde und damit verbunden die Aminosäure und die Energiedifferenz angegeben ist:

```
String strDetails = m.getDetailList();
```

Die Einträge dieser Liste sind mit Zeilenumbrüchen (`\r\n`), und die Informationen innerhalb eines Eintrags mittels Tabstopp getrennt.

```
Distanz     Aminosäure  Energiedifferenz
0.0         84:A:L      15.03
3.8         83:D:D      1.358
3.822       85:A:A      1.08
4.397       89:G:G      0.533
4.893       90:T:T      1.08
5.379       82:G:G      0.533
5.59        86:L:L      0.533
5.681       121:Q:Q     2.927
```

Weiterhin besteht die Möglichkeit, die Informationen aus der Detailliste verarbeitet in einem Abstandsdiagramm zu erhalten. Dieses Diagramm ist in Form eines „BufferedImage" verfügbar:

```
BufferedImage imgMutation = m.getImage();
```

Standardmäßig wird dieses quadratische Diagramm mit einer Größe von 400 mal 400 Pixeln erzeugt. Wenn eine andere Größe gewünscht wird, kann dies mit folgender Methode erreicht werden:

```
BufferedImage imgMutation = m.getImage(nSize);
```

4.3.2. Funktionsweise

Die eigentliche Einstiegsfunktion für „eMut" ist die Methode „setData", da es für den Konstruktor keine zu initialisierenden Daten gibt. Die Methode setzt dabei die übergebenen Energiedaten und resettet die beiden Ausgabewerte.

```
public void setData(EData data1, EData data2)
{
      eData1 = data1;               // Energiedaten übernehmen
      eData2 = data2;

      strSequence = null;           // Ausgabewerte resetten (1)
      nMutations  = 0;
}
```

Durch das Zurücksetzen der Ausgabewerte wird der Ausgangszustand für „eMut" hergestellt, die neuen Energiedaten dienen der Verarbeitung für die entsprechenden Abfragen. Wichtig ist, dass diese Funktion vor allen anderen bei der Benutzung von „eMut" aufgerufen wird.

Sowohl „getSequenceMutation" als auch „getMutations" folgen dem gleichen Aufbau. Solange die Ausgabewerte den Standardwerten entsprechen (die Standardwerte entsprechen den bei (1) angegebenen Werten) wird bei einem Aufruf der jeweiligen Methode zuerst der Ausgabewert gefüllt. Sollte der Ausgabewert vom Standardwert abweichen, so wird einfach nur der Ausgabewert zurückgeliefert. Auf diese Weise kann die Ausgabe mehrfach angefordert und muss nur einmal berechnet werden.

Die Methode „getSequenceMutation" analysiert die Energiedaten, indem es die Aminosäuren Stelle für Stelle in einer Schleife durchläuft. Falls Sequenzungleichheiten auftreten wird an dieser Stelle eine Mutation vermutet und die entsprechenden Einträge im Rückgabearray werden bearbeitet. Zusätzlich wird die Anzahl der Mutationen (`nMutations`) mitgezählt und auf den entsprechenden Wert gesetzt. Die Sekundärstruktur wird auf gleiche Weise wie die Sequenz durchlaufen und bei Ungleichheiten in dieser werden die entsprechenden Vermerke im Rückgabearray vorgenommen. Der Durchlauf durch die Sequenz ermöglicht es auch die restlichen Einträge im Array zu füllen (Sequenzen an sich und Energiedifferenzen).

Das Abrufen der Mutationen erfolgt im Gegensatz zu den anderen beiden Funktionen On-the-fly. Es wird also jedesmal eine neue Berechnung durchgeführt, um die Informationen der Mutation an der entsprechenden Stelle zu erhalten. Es wird die Mutation mit dem entsprechenden Index herausgesucht und ein neues „Mutation"-Objekt erzeugt. Diesem wird die Aminosäure aus dem ersten Datensatz als Mutationsstelle über den Konstruktor mitgegeben. Ist der Index gültig, wurde also eine Mutation gefunden, bevor die Sequenz komplett durchlaufen wurde, werden für alle Einträge in den Datensätzen die Energiedifferenzen berechnet. Sollte die berechnete Differenz zweier Werte von 0 abweichen, so wird diese Stelle der Mutation hinzugefügt. Nachdem der Durchlauf beendet wurde, wird das „Mutation"-Objekt zurück geliefert oder `null`, falls ein ungültiger Index angegeben wurde.

Die Klasse „Mutation" nimmt in ihrem Konstruktor die Mutationsstelle in Form eines „Atom"-Objekts entgegen und initialisiert eine Liste für die einzelnen Mutationseinträge. Diese Mutationseinträge werden durch einen Container (`MutationEntry`) gekapselt. Jeder Eintrag enthält dabei den Abstand zur Mutationsstelle, die Namen der aktuellen Aminosäuren (aus beiden Energiedaten), sowie die Energiedifferenz.

Um einen solchen Mutationseintrag zur Mutation hinzuzufügen steht folgende Methode bereit:

```
m.addEntry(eData1.getAtom(i), eData2.getAtom(i), dEnergyDifference);
```

Dabei wird die Distanz zwischen der Mutationsstelle und dem ersten Parameter, also der Aminosäure aus dem ersten Energiedatensatz berechnet. Da die Positionsangaben in den PDB-Dateien mitunter stark unterschiedlich sind und anscheinend keinem direkten Muster folgen, wurde als Bezugspunkt immer der erste Energiedatensatz gewählt (auch bei der Übergabe der Mutationsstelle im Konstruktor der „Mutation"-Klasse). Nachdem die Berechnung der Distanz durch die Differenzenbildung der Raumkoordinaten abgeschlossen ist, wird sie mit den restlichen Informationen in einen Container überführt und an die Liste der Mutationseinträge angehangen.

Bei der Abfrage der detailierten Liste über die Mutation, wird als erstes die Liste mit den Mutationseinträgen nach Abstand zur Mutationsstelle sortiert. Anschließend werden die einzelnen Einträge mit Hilfe eines StringBuilders aufbereitet. Der Ausgabestring dieser Methode wird, wie auch (1), nur einmalig zusammengestellt und kann anschließend beliebig oft ohne weiteren Aufwand empfangen werden. Dabei ist zu beachten, dass nachdem diese Methode aufgerufen wurde, keine weiteren Mutationseinträge hinzugefügt werden können.

Für die visuelle Ausgabe der Mutation als Diagramm wird eine ähnliche Vorgehensweise benutzt. Der Unterschied liegt darin, dass die Sortierung aufsteigend nach den Energiedifferenzen erfolgt. Der Vorteil liegt darin, dass die größten Differenzen zuletzt in das Diagramm eingezeichnet werden. Warum dies ein Vorteil ist, liegt in der Art, wie die Diagramme erzeugt werden. Es wird jeder Mutationseintrag durchlaufen und anhand des Abstandes zur Mutationsstelle ein Ring mit einer bestimmten Färbung eingetragen. Die Färbung orientiert sich dabei an der Größe der Energiedifferenz. Abhängig von der größten Energiedifferenz wird eine Abstufung in drei gleiche Bereiche vorgenommen. Der untere Bereich erhält die Farbe Grün, der mittlere Bereich erhält einen Orangeton und die größten Energiedifferenzen werden Rot markiert. Durch die Sortierung nach Energiedifferenzen werden die hohen Werte nicht von den niedrigeren Werten überdeckt. Um das Diagramm noch ein wenig übersichtlicher zu gestalten, wird am oberen Rand der entstehenden Scheibe der größte Abstand als Radius, sowie aller 8 Å ein schwarzer Orientierungskreis eingetragen.

Die Sortierungen der Liste mit Mutationseinträgen wird über die von Java bereitgestellte Sortierfunktion für „Collections" vorgenommen, indem ein eigener Comparator definiert wird:

```
Collections.sort(listEntries, new Comparator<MutationEntry>() {
@Override public int compare(MutationEntry e1, MutationEntry e2)
{
      int nReturn;
      double dDistance = e1.getDistance() - e2.getDistance();

      if (dDistance < 0)      { nReturn = -1; }
      else if (dDistance > 0) { nReturn = 1; }
      else                    { nReturn = 0; }

      return nReturn;
}});
```

Dabei steht dieser Ausschnitt für die Sortierung anhand der Abstände zur Mutationsstelle. Die Sortierung anhand der Energiedifferenzen folgt aber dem gleichen Muster.

4.4. eGor

Das Tool „eGor" (energy gor-prediction) dient der Vorhersage von Energieprofilen anhand von Sequenzen und basiert auf dem GOR-Algorithmus. Dieses Tool ist in der Projektgruppe für die bioinformatischen Tools entstanden und wurde nur in den Server integriert. Die genauere Funktionsweise kann in der Literatur [1] nachgeschlagen werden.

Die Benutzung erfolgt nach folgendem Schema:

```
// Instanziierung der Klasse
EGor eGor = new EGor(strSequence, bTm);
```

Bei der Instanziierung der Klasse werden dem Konstruktor zwei Parameter übergeben. Zum Einen ist das die Sequenz (`strSequence`) und zum Anderen ein boolscher Wert (`bTm`). Er gibt an, ob die Sequenz von einem transmembranen (`true`) oder einem nicht-transmembranen (`false`) Protein stammt. Dies ist ausschlaggebend für die Vorhersage des Energieprofils. Die Sequenz muss dabei mindestens 7 Aminosäuren umfassen, da andernfalls der Vorhersagealgorithmus scheitert. „eGor" beginnt innerhalb des Konstruktors direkt mit der Vorhersage.

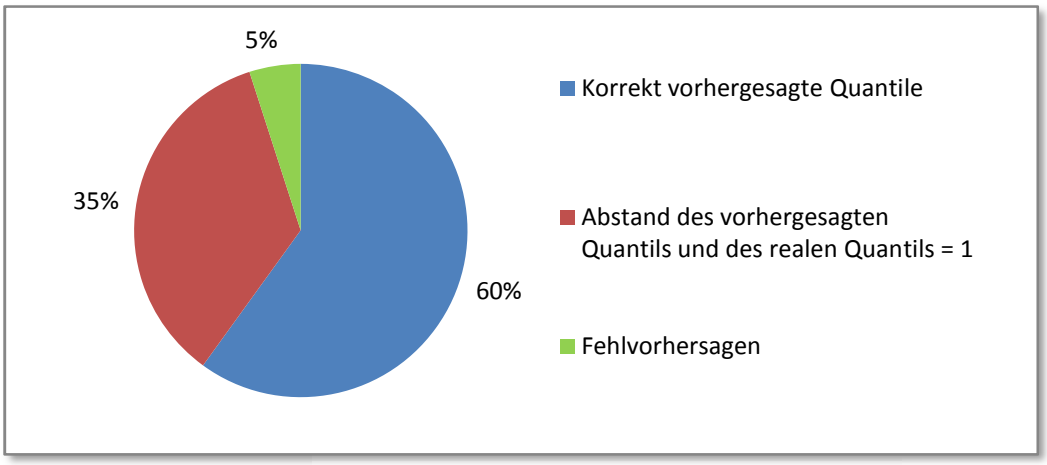
Folgende Funktionen stehen bereit um die Ergebnisse zu erhalten:

```
String strSequence            = eGor.getPredictedSequence();
ArrayList<Double> listEnergy  = eGor.getPredictedEnergyList();
```

Das vorhergesagte Energieprofil ist um 6 Aminosäuren kürzer als die eigentliche Eingabesequenz. Es werden am Anfang, sowie auch am Ende der Sequenz je 3 Aminosäuren abgeschnitten, damit der Vorhersagealgorithmus funktionieren kann. Der String `strSequence` beinhaltet diese neue vorhergesagte (gekürzte) Sequenz.

In der ArrayList `listEnergy` ist das vorhergesagte Energieprofil (zumindest die Energiewerte) zu finden. Diese Werte beziehen sich auf die vier Quantile, entsprechend der Einstellungen, ob es sich bei dieser Sequenz um einen transmembranen oder einen nicht-transmembranen Bereich handelt.

Vorhergesagte Energieprofile können also zum aktuellen Entwicklungsstand nur die eventuellen Energiequantile zu den entsprechenden Aminosäuren zuordnen. Dabei ist die Vorhersagegenauigkeit recht zuverlässig.



4.5. eAlign

Wie auch „eGor" ist „eAlign" (energy alignment) in der Projektgruppe für die bioinformatischen Tools entstanden und wurde nur in den Server integriert. Es liefert einen Algorithmus zum paarweisen Vergleich zweier Energieprofile auf Basis des Needleman-Wunsch-Algorithmus. Die Funktionsweise des „eAlign" kann in der Literatur [1] nachgeschlagen werden.

4.5.1. Nutzung

Die Benutzung erfolgt nach folgendem Muster:

```
// Instanziierung der Klasse
EAlign eAlign = new EAlign();

// Optionale Einstellungen vornehmen
eAlign.setGapCost(-15);                  // Standardwert: -30
eAlign.setGapExtend(-3);                 // Standardwert: -7
eAlign.setGlobalAlignment(false);        // Standardwert: true
eAlign.setNumberOfPermutations(500);     // Standardwert: 50

// Alignment starten
eAlign.setData(eData[0], eData[1]);      // Energiedaten übergeben
eAlign.processAlignment();               // Alignment durchführen
```

Nun kommt es auf die gewünschten Informationen an, mit denen man arbeiten möchte. Die erste Möglichkeit bietet das Energiealignment selbst und der berechnete Z-Score.

```
String strEAlignment = eAlign.getEnergyAlignment();
double dZScore = eAlign.getZScore();
```

Das Energiealignment liegt in dem String `strEAlignment` in folgender Form, von Zeilenumbrüchen (`\r\n`) getrennt, vor:

```
Alignierte Sequenz 1
Zeile mit dem Alignment
Alignierte Sequenz 2
```

(Wörtliche Beschreibung)

```
RSIKAICENKNGNPHRENLRISKSSFQVTT
 |':|.|    :|'|.  ||||||:|.|:
-QDRFHL----TEVHS--LNVLEGSWVLY-
```

(Als Beispiel ein Ausschnitt)

Die in der Alignment-Zeile aufgeführten Zeichen haben folgende Bedeutungen:

- | - Die Energiewerte liegen im selben Quantil
- : - Die Energiewerte liegen in einem benachbartem Quantil
- . - Die Energiewerte liegen zwei Quantile auseinander
- ' - Die Energiewerte liegen drei Quantile auseinander
- ~ - Die Energiewerte haben keinen Zusammenhang mehr
- Ein Leerzeichen wird eingefügt, wenn Lücken (`-`) auftreten

Der zScore (`dZScore`) bildet einen genormten Wert, welcher die Übereinstimmung zwischen zwei Energieprofilen darstellt. Man kann diesen Wert grob in drei unterschiedliche Bereiche einteilen:

- Bereich < 1: Die beiden Energieprofile haben keine signifikanten Ähnlichkeiten
- Bereich 1 <= zScore < 2: Die Energieprofile besitzen einige Ähnlichkeiten
- Bereich > 2: Die Energieprofile besitzen viele Ähnlichkeiten

Daher gilt, je höher der erzielte zScore, desto größer ist die vorhandene Übereinstimmung und desto ähnlicher sind sich die beiden Energieprofile.

Ein weiterer verwendeter Komplex, welcher „eAlign" zur Verfügung stellt, ist die Erzeugung eines Dotplots. Dieser kann mittels der folgenden Funktionen rekonstruiert werden:

```
char[][] cMatrixPath    = eAlign.getPathMatrix();
int[][] nMatrix         = eAlign.getDotplotMatrix();
int[] nLocalPos         = eAlign.getPositionOfHighestValue();
```

Die Pfadmatrix `cMatrixPath` beinhaltet den zur Berechnung des Alignments genommenen Weg. Die eigentliche Wert-Matrix, anhand der das Alignment ausgeführt wird, ist daher nicht mehr notwendig. Mit Hilfe der Pfadmatrix ist es möglich, den genauen Pfad der Berechnung in der Matrix für die Werte des Dotplots (`nMatrix`) zu verfolgen. Dies wird von „eProS" verwendet und ist als grauer Pfad im Dotplot wahrzunehmen. Falls es sich um ein lokales Alignment handelt, soll nicht der am weitesten rechts unten stehende Wert als Ausgangspunkt genommen werden, sondern der größte Wert innerhalb der Matrix. Die Position des größten Wertes kann in `nLocalPos` erfragt werden.

Bei Positionsangaben generell ist zu beachten, dass „eAlign" stets die ursprünglich längere Sequenz auf der Horizontalen und die kürzere Sequenz auf der Vertikalen abbildet.

4.5.2. eSearch

Das Tool „eSearch" (energy search tool) sucht auf der Grundlage von „eAlign" für ein vorgegebenes Protein oder eine Sequenz das am besten energetisch übereinstimmende Protein aus dem Datensatz.

Dabei durchsucht „eSearch" den gesamten Datensatz und berechnet mit Hilfe von „eAlign" den zScore zwischen dem zu untersuchenden Protein und jedem gespeicherten Protein im Datensatz. Die Besten dieser Werte werden in eine Liste eingetragen, die ständig aktualisiert wird. Am Ende des Durchlaufs liegt eine Liste mit den besten zScores, oder besser formuliert, mit den größten energetischen Ähnlichkeiten, zwischen einem vorgegebenen Protein und dem Datensatz vor.

Ein Durchlauf durch den gesamten Datensatz ist zeitintensiv. Aus diesem Grund wurde sich dazu entschieden bei Abschluss des Vorgangs eine Benachrichtigung an eine eMail-Adresse zu versenden, welche hinterlegt werden kann. Der Empfänger der eMail erhält eine eindeutige ID und einen Link mit dessen Hilfe er sich die berechneten Ergebnisse betrachten kann.

5. Servlets

Ein Servlet ist im eigentlichen Sinn eine normale Java-Klasse, die von der Klasse „HttpServlet" abgeleitet ist und das Interface „Servlet" implementiert. Damit ein Servlet über den Web-Browser ansprechbar ist, muss es als erstes in den Deployment Descriptor eingetragen werden. Dazu sind folgende Einträge notwendig.

```
<servlet>
      <description>Returns a requested file.</description>
      <display-name>GetFile</display-name>
      <servlet-name>GetFile</servlet-name>
      <servlet-class>
            de.hsmw.bioinformatics.epros.servlet.GetFile
      </servlet-class>
</servlet>
```

Der erste Eintrag registriert das Servlet und beschreibt dabei seine Eigenschaften. Darunter fällt eine Kurzbeschreibung des Servlets (`description`), der Anzeigename (`display-name`), der Servletname (`servlet-name`) und der Klassenname inklusive Packageangabe (`servlet-class`).

```
<servlet-mapping>
      <servlet-name>GetFile</servlet-name>
      <url-pattern>/GetFile</url-pattern>
</servlet-mapping>
```

Der zweite notwendige Eintrag bildet eine Zuordnung zwischen dem Servletname und einem URL-Muster. Über den im „servlet"-Tag gewählten Servletname ist festgelegt, um welches Servlet es sich genau handelt.

Sobald eine Anfrage an den Web-Server gestellt wird, werden alle URL-Muster mit der URL der Anfrage abgeglichen. Sollte ein Muster passen, so wird die Anfrage an das entsprechende Servlet weitergeleitet.

5.1. Hierarchie

5.1.1. Packages

Alle entwickelten Tools und Anwendungen befinden sich im Package „de.hsmw.bioinformatics". Diese Domain soll die Grundlage für eine zusammengehörige Bibliothek liefern.

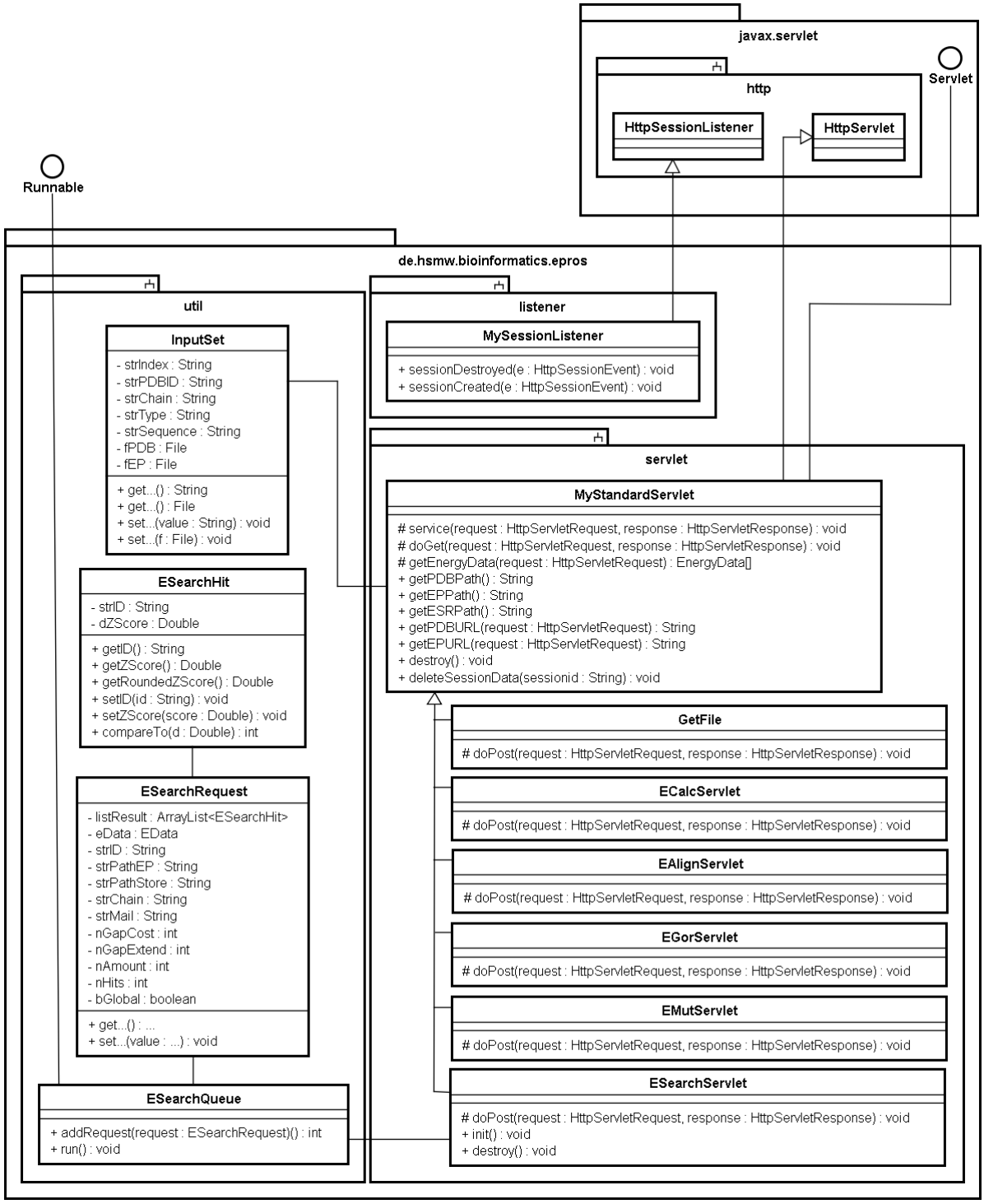
„eProS" hat sich in dieser Package-Hierarchie mit „de.hsmw.bioinformatics.epros" etabliert. Dabei entstanden noch die Subpackages „listener", „util" und „servlet". Damit ist die Unterteilung in die einzelnen Aufgabenbereiche spezifiziert.

Im Package „listener" ist der SessionListener zu finden, der sich um die „Aufräumarbeiten" kümmert, nachdem eine Session beendet wurde.

Das Package „servlet" beinhaltet die derzeit 6 von außen ansprechbare Servlets, sowie die Oberklasse „MyStandardServlet", welches alle Grundfunktionen bereitstellt.

Im „util"-Package liegen die Containerklassen und andere Hilfsklassen, die von den Servlets benötigt werden.

5.1.2. Klassendiagramm



Klassendiagramm mit allen Servlets, Util-Klassen und dem Listener für „eProS"

Das Klassendiagramm bildet alle öffentlichen (public) und geschützten (protected) Methoden ab. Getter- und Setter-Methoden werden bei größerem Umfang in den Util-Klassen abgeschnitten und dafür die privaten Attribute abgebildet. Auf Details wurde verzichtet.

5.2. listener.MySessionListener

Im Gegensatz zu den Servlets kann ein „SessionListener" nicht direkt angesprochen werden. Es ist damit weder ein Name noch ein Mapping notwendig.

```
<listener>
      <listener-class>
            de.hsmw.bioinformatics.epros.listener.MySessionListener
      </listener-class>
</listener>
```

Es ist ebenfalls nicht möglich einem Listener Initialisierungsparameter mitzugeben. Der Deployment Descriptor ermöglicht es aber verschiedene Einstellungen mittels weiterer Tags vorzunehmen. So kann über folgenden Eintrag die Dauer einer Session in Minuten festgelegt werden:

```
<session-config>
      <session-timeout>720</session-timeout>
</session-config>
```

Die erste Aufgabe, die der „MySessionListener„ für „eProS" erfüllt, ist das „Aufräumen", nachdem eine Session beendet wurde. Dabei werden gespeicherte Session-Daten gelöscht um keinen unnötigen Datenmüll anzuhäufen. Die Session-Daten beschreiben dabei die abgelegten PDB- und EP-Dateien in den jeweiligen Ordnern.

Die zweite Aufgabe ist es, veraltete ESR-Dateien zu löschen. Als veraltet zählen dabei alle berechneten Anfragen, die älter als eine Woche sind.

5.3. servlet.MyStandardServlet

Das Servlet „MyStandardServlet" bildet die Oberklasse für alle Servlets, die „eProS" anbietet. Es ist dabei nicht direkt ansprechbar, besitzt also demnach keinen Eintrag im Deployment Descriptor.

Es bietet zahlreiche Standardfunktionen an, die von allen Servlets genutzt werden und daher nicht jedesmal neu implementiert werden müssen. Darunter fallen die Initialisierungen der Pfade zu den Datenordnern und das Verarbeiten der Eingabedaten, die über den HTTP-Request gesendet werden.

5.3.1. Funktionsweise

Da ein Servlet-Objekt nur einmal erzeugt wird und es damit nur eine Instanz des Servlets existiert, wird jeder Request in einem eigenen Thread aufgerufen. Die daraus resultierenden Thread-Problematiken sollten dabei immer im Hintergrund bedacht werden.

Für jede Anfrage eines Clients, wird die service-Methode des entsprechenden Servlets aufgerufen. Diese sollte jedoch nicht überschrieben werden, da dadurch das korrekte Handling der einzelnen HTTP-Methoden (GET, POST, PUT, …) nicht mehr gewährleistet ist. Die service-Methode leitet die Anfrage je nach HTTP-Methode an die entsprechende Funktion weiter. So wird beispielsweise ein POST-Request an die doPost-Methode und ein GET-Request an die

doGet-Methode weitergeleitet. Diese beiden Methoden sind zum Abarbeiten der Anfragen wesentlich besser geeignet. Eine Anfrage wird in Form eines „HttpServletRequest"-Objekts an die jeweilige Methode übergeben.

Die service-Methode wird im „MyStandardServlet" trotzdem überschrieben. In dieser Funktion werden zusätzlich die Pfade, sowie die URL-Pfade für die Applets, zu den einzelnen Dateiordnern in der Session abgelegt. Anschließend wird die service-Methode der Oberklasse aufgerufen um den normalen Ablauf nicht zu unterbrechen. Das Ablegen der Ordnerpfade ist notwendig für die in der JSP aufzubereitenden Daten.

```
request.getSession().setAttribute("pathPDB",    getPDBPath());
request.getSession().setAttribute("pathEP",     getEPPath());
request.getSession().setAttribute("pathESR",    getESRPath());

request.getSession().setAttribute("urlPDB",     getPDBURL(request));
request.getSession().setAttribute("urlEP",      getEPURL(request));

super.service(request, response);
```

Die doGet-Methode wird ebenfalls im „MyStandardServlet" überschrieben. Sie leitet jetzt alle Anfragen an die doPost-Methode weiter. Daraus folgt, dass alle Anfragen von der doPost-Methode bearbeitet werden und nur diese überschrieben werden muss.

Das Servlet überschreibt nicht nur Methoden der Oberklasse, es bietet auch neue Funktionen für die „eProS"-Servlets an. Die ersten zu erwähnenden Methoden wurden im vorherigen Codeausschnitt schon vorgestellt. Sie dienen dazu, die Ordnerpfade unabhängig vom unterliegenden Web-Server den Servlets und den JSP-Seiten zur Verfügung zu stellen.

```
// Lieferen die Pfade zu den entsprechenden Datenordnern
getPDBPath();
getEPPath();
getESRPath();
// Liefern die URL-Pfade zu den entsprechenden Datenordnern
getPDBURL(request);
getEPURL(request);
```

Ein Pfad wird anhand eines festgelegten Strings, der die Position des jeweiligen Ordners relativ angibt, und dem „ServletContext", welcher die absolute Komponente darstellt, zusammengefügt. Dies erfolgt bei allen Methoden nach dem gleichen Muster:

```
// Relative Pfadangabe zu den Datenordnern
private final String PATH_PDB = "/EprosData/pdb/";
private final String PATH_EP  = "/EprosData/ep/";
private final String PATH_ESR = "/EprosData/esr/";

// Aufbau eines Pfades am Beispiel des PDB-Pfades
public String getPDBPath()
{
String strPath = getServletContext().getRealPath("");
strPath = strPath.substring(0, strPath.lastIndexOf(File.separator));
strPath = strPath + PATH_PDB.replace('/', File.separatorChar);
return strPath;
}
// Aufbau einer URL am Beispiel der PDB-URL
public String getPDBURL(HttpServletRequest request)
{ return    "http://" + request.getServerName() + ":" +
            request.getLocalPort() + PATH_PDB;             }
```

Die zweite Funktion, die dieses Servlet anbietet, dient dazu, die bei der Anfrage an das Servlet gestellten Daten auszuwerten. Dazu zählt das Auslesen und die Validierung der übergebenen Parameter, sowie die Aufbereitung der Daten für die Servlets.

```
EData[] eData = getEnergyData(request);
```

Die Funktionsweise dieser Methode wird im folgenden Kapitel genauer beschrieben. Falls in dieser Methode eine Eingabe zu ungültigen Daten führt, wird ein entsprechendes Attribut für die Anfrage gesetzt, die den genauen Fehler in der Eingabemaske markiert.

5.3.2. Datenübergabe

Um im Servlet einen Feldwert aus einem abgeschickten Formular zu erhalten, werden die Daten aus der Anfrage, dem „HttpServletRequest"-Objekt (`request`), benötigt. Dies ist über die getParameter-Methode möglich. Die Methode erwartet dabei als Parameter einen String, welcher den Namen des Feldes repräsentiert und liefert einen String mit dem Feldwert.

```
// Simples Formular mit einem Textfeld
<form ...>
      PDBID: <input type="text" name="pdbid">
</form>

// Abfrage des Feldwertes im Servlet
String strPDBID = request.getParameter(„pdbid");
```

Es existieren verschiedene HTTP-Methoden (POST, GET, HEAD, …) Daten an ein Servlet zu übergeben. Die für „eProS" relevanten sind dabei nur die POST- und die GET-Methode. Bei der Übergabe bzw. beim Absenden von Formulardaten, kann die entsprechende Methode im form-Tag der HTML-Anweisung angegeben werden (`method="POST"`). Alle in „eProS" vorhandenen Formulare übergeben ihre Parameter mit der POST-Methode. Zusätzlich ist es möglich, die Art der Übertragung genauer festzulegen. Dazu dient das enctype-Attribut im form-Tag. Da dem Nutzer angeboten wird seine eigenen Daten auf den Server hochzuladen, muss das enctype-Attribut den Wert „multipart/form-data" erhalten. Somit ist es möglich einen File-Upload zu realisieren (Das type-Attribut im input-Tag des Formulars muss den Wert „file" erhalten).

Durch die soeben aufgeführten Einstellungen, können die übergebenen Feldwerte nicht mehr über die getParameter-Methode abgefragt werden. Durch die Angabe des enctype-Attributs mit dem Wert „multipart/form-data", werden die übergebenen Formulardaten nun auf eine andere Art im HTTP-Header codiert. Die Daten können ausgelesen werden, indem man den HTTP-Header nach den gewünschten Informationen durchsucht.

```
ServletInputStream streamInput = request.getInputStream(); // (1)
byte[] byteLine   = new byte[128];
int nRead         = streamInput.readLine(byteLine, 0, 128);
```

Der HTTP-Header ist in verschiedene Bereiche eingeteilt. Die Parameter und übergebenen Daten sind im Content-Bereich zu finden, auf dem sich im Folgenden bezogen wird. Mit Hilfe von (1) kann man den Content-Bereich auslesen.

Die einzelnen Feldwerte werden von einem speziellen String voneinander abgegrenzt. Dieser Abgrenzungs-String (`strBoundary`) kann ermittelt werden, indem die erste Zeile des Content-Bereichs gelesen wird. Anhand dieser Grenze können nun die einzelnen Abschnitte mit den

Informationen durchsucht werden. Das Ende des Content-Bereichs wird durch den Abgrenzungs-String mit zwei zusätzlichen Bindestrichen am Ende gekennzeichnet.

Für normale Formularfelder, ist nach dem Abgrenzungs-String diese Zeile zu finden:

```
Content-Disposition: form-data; name="inputName"
```

Dabei steht `inputName` für den Feldname. An die Zeile schließt sich eine Leerzeile und der Feldwert an:

```
-----------------------------7d15340138        // Abgrenzungs-String
Content-Disposition: form-data; name="pdbid"   // Feldname
                                               // Leerzeile
1A3H                                           // Feldwert
```

Für Datei-Upload-Felder enthält die erste Zeile nach dem Abgrenzungs-String zusätzliche Informationen über den Dateinamen. Weiterhin folgt noch eine Zeile, in welcher der Dateityp näher beschrieben wird, bevor die Leerzeile eingefügt wird.

```
Content-Disposition: form-data; name="pdbfile"; filename="1a3h.pdb"
Content-Type: text/plain
```

Der Inhalt der Datei folgt genau so, wie bei den normalen Formularfeldern, nach der Leerzeile und bis zum nächsten Abgrenzungs-String. Die Datei kann somit aus dem Content ausgelesen und gespeichert werden.

Der komplette Content-Bereich des HTTP-Headers kann auf diese Weise durchlaufen und geparst werden. Da „eProS" auch GET-Anfragen unterstützt, muss an dieser Stelle aber ein Schalter eingebaut werden, um Fehler zu vermeiden. Normale GET-Anfragen besitzen keinen so komplizierten Header. Sie lassen sich über die getParameter-Methode des „HttpServletRequest"-Objekts auslesen. Die einfachste Möglichkeit diesen Schalter zu realisieren, ist die Abfrage der Größe des Abgrenzungs-Strings. Ist dieser kleiner als drei Zeichen groß, so handelt es sich um eine normale Anfrage, ansonsten liegt eine „multipart/form-data"-Anfrage vor.

Es liegen zwei Fälle vor, in denen die „eProS"-Servlets über GET-Anfragen angesteuert werden und nicht über Formulare. Der erste Fall tritt ein, wenn die JSP-Seite in ihren Ausgangszustand zurückgesetzt werden soll. Der zweite Fall tritt ein, wenn eine Ergebnis-ID von „eSearch" als direkter Link zum „ESearchServlet" durchgereicht werden soll.

Unabhängig von der Art, wie die Daten ausgelesen werden, werden zusammengehörige Daten anhand eines Index im Feldname, in eine Container-Klasse („InputSet") gepackt. Werte, die nicht für dieses „InputSet" vorgesehen sind, werden als Attribute im Request-Objekt hinterlegt. Um Werte zwischen Servlets und JSP-Seiten zu übermitteln, kann man sie als Attribute im Request- oder auch im Session-Objekt ablegen:

```
// Ein Attribut für den aktuellen Request setzen
request.setAttribute(strName, oValue);
// Ein Attribut für die aktuelle Session setzen
request.getSession().setAttribute(strName, oValue);
```

Die setAttribute-Methode erwartet zwei Parameter. Der erste Parameter ist ein String (`strName`) und spezifiziert einen eindeutigen Namen. Der zweite Parameter ist vom Typ Object (`oValue`) und stellt den Wert für das Attribut dar. Ein Attribut existiert so lange, wie der Container (Request oder Session), dem man es zuordnet.

Nachdem die Informationen aus der Anfrage verarbeitet wurden, liegen je nach Anfrage zwischen null und zwei „InputSet"-Objekte vor. Auf Grundlage ihrer Daten werden demnach zwischen null und zwei „EData"-Objekte geladen und dem Servlet bereit gestellt.

Das „InputSet" (`set`) stellt dazu folgende Informationen zur Verfügung, die beim Parsen der Anfrage eingelesen wurden:

```
// Fester, eindeutiger Index
String strIndex         = set.getIndex();       // Index des Set

// Eigenschaften
String strChain         = set.getChain();       // Ketten-Bezeichner
String strType          = set.getType();        // Typ des Proteins

// Nur einer dieser Werte wird betrachtet (nur eine Eingabe möglich)
String strPDBID         = set.getPDBID();       // PDB-ID
File fPDB               = set.getPDBFile();     // eigene PDB-Datei
File fEP                = set.getEPFile();      // eigene EP-Datei
String strSequence      = set.getSequence();    // Sequenz
```

Der Index (`strIndex`) ist nur für Zuordnungszwecke von Bedeutung. Damit wird sichergestellt, dass Attribute, die nicht mit im „InputSet" abgelegt wurden, und denselben Index haben wie die Daten im Set trotzdem richtig im Servlet zugeordnet werden können. Die Informationen zu einer Kette (`strChain`) liefern den Ketten-Bezeichnet oder einen leeren String, falls keine Kette näher betrachtet werden soll. Der Typ des Proteins (`strType`) hat entweder den Wert „TM" (transmembranes Protein) oder „nTM" (nicht-transmembranes Protein). Die vier möglichen Eingabedaten werden in der Reihenfolge, wie oben angegeben, auf einen Inhalt geprüft. Der erste Wert, der einen Inhalt besitzt wird als Eingabe-Wert genutzt. Wenn es sich dabei um eine PDB-ID handelt (`strPDBID`), wird als erstes geprüft, ob zu dieser ID Daten im aktuellen Session-Ordner vorhanden sind. Wenn das nicht der Fall ist, wird im Datensatz nach dieser ID geschaut. Wenn die ID weder im Session-Ordner, noch im Datensatz vorhanden ist, wird sie von der Seite der www.pdb.org in den Session-Ordner heruntergeladen und anhand der Eigenschaften (`strChain` und `strType`) wird das Energieprofil berechnet. Wenn eine eigene PDB-Datei (`fPDB`) hochgeladen wurde, wird auf dem gleichen Weg vorgegangen, als währe diese Datei soeben heruntergeladen wurden. Die dritte Eingabemöglichkeit ist das Hochladen einer eigenen EP-Datei (`fEP`). Diese wird anhand ihrer Beschaffenheit als einfache *.ep-Datei oder als detailierte *.ep2-Datei identifiziert und aus den gewonnen Informationen wird ein „EData"-Objekt erzeugt. In dieser Variante wird der Typ des Proteins nicht beachtet, da das Protein bereits berechnet wurde. Wenn möglich, wird eine PDB-Datei zu diesem Energieprofil gesucht. Wenn der Nutzer einer Sequenz (`strSequence`) als Eingabe gewählt hat, so wird diese mit Hilfe von „eGor" in ein vorhergesagtes Energieprofil überführt. Die Angabe einer Kette wird hierbei nicht beachtet. Das entstandene oder geladene Energieprofil wird als „EData"-Objekt zurückgeliefert.

5.4. servlet.ECalcServlet

Das „ECalcServlet" dient der Darstellung eines Energieprofils und des dazugehörigen Proteins.

Neben einem „EData"-Objekt werden folgende Attribute benötigt um die Ausgabedaten aufzubereiten. Die aus der Anfrage bezogen Attribute:

Request-Attribut	Typ	Beschreibung
chain0	String	Die darzustellende Kette
reset (Parameter)	Ohne Typ	Setzt alle eCalc-Session-Attribute zurück

Um die entsprechenden Daten in der zugehörigen JSP („eCalc.jsp") darzustellen, werden diese Session-Attribute gesetzt:

Session-Attribut	Typ	Beschreibung
eCalc	Boolean	Schalter, zwischen der Eingabemaske und der aufbereiteten Ausgabe (null = Eingabemaske)
eCalc_chain	String	Die darzustellende Kette
eCalc_pdbid	String	Die darzustellende PDB-ID (EP-ID)
eCalc_pdbnotes	String	Informationen über das Energieprofil

5.5. servlet.EAlignServlet

Um mit dem „EAlignServlet" die Daten für ein Energie-Alignment aufzubereiten, wird das Tool „eAlign" genutzt.

Request-Attribut	Typ	Beschreibung
chain0	String	Die darzustellende Kette für Datensatz 0
chain1	String	Die darzustellende Kette für Datensatz 1
gapcost	Integer	Kosten für eine neue Lücke im „eAlign"
gapextend	Integer	Kosten für das erweitern einer Lücke im „eAlign"
method	String	Methode für „eAlign" (lokal oder global)
reset (Parameter)	Ohne Typ	Setzt alle eAlign-Session-Attribute zurück

Die berechneten Ergebnisse werden an die JSP „eAlign.jsp" weitergeleitet:

Session-Attribut	Typ	Beschreibung
eAlign	Boolean	Schalter, zwischen der Eingabemaske und der aufbereiteten Ausgabe (null = Eingabemaske)
eAlign_alignment	String	Alignment-String der Datensätze
eAlign_chain0	String	Die darzustellende Kette für Datensatz 0
eAlign_chain1	String	Die darzustellende Kette für Datensatz 1
eAlign_evis_alignment	String	Alignment-String der Datensätze für „eVis"
eAlign_pdbid0	String	Die darzustellende PDB-ID (EP-ID) mit Index 0
eAlign_pdbid1	String	Die darzustellende PDB-ID (EP-ID) mit Index 1
eAlign_zscore	Double	Der berechnete zScore

5.6. servlet.EGorServlet

Die Aufgabe des „EGorServlet" besteht darin, das vorhergesagte Energieprofil darzustellen. Die Nutzung des Tools erfolgte bereits beim Parsen der Eingabedaten.

Request-Attribut	Typ	Beschreibung
reset (Parameter)	Ohne Typ	Setzt alle eGor-Session-Attribute zurück

Die berechneten Ergebnisse werden an die JSP „eGor.jsp" weitergeleitet:

Session-Attribut	Typ	Beschreibung
eGor	Boolean	Schalter, zwischen der Eingabemaske und der aufbereiteten Ausgabe (null = Eingabemaske)
eGor_pdbid	String	Die darzustellende PDB-ID (EP-ID)
eGor_sequence	String	Die Ausgangssequenz

5.7. servlet.EMutServlet

Das „EMutServlet" bereitet die Ausgabedaten für Punktmutationen mit Hilfe des Tools „eMut" auf.

Request-Attribut	Typ	Beschreibung
chain0	String	Die darzustellende Kette für Datensatz 0
chain1	String	Die darzustellende Kette für Datensatz 1
reset (Parameter)	Ohne Typ	Setzt alle eMut-Session-Attribute zurück

Die berechneten Ergebnisse werden an die JSP „eMut.jsp" weitergeleitet:

Session-Attribut	Typ	Beschreibung
eMut	Boolean	Schalter, zwischen der Eingabemaske und der aufbereiteten Ausgabe (null = Eingabemaske)
eMut_chain0	String	Die darzustellende Kette für Datensatz 0
eMut_chain1	String	Die darzustellende Kette für Datensatz 1
eMut_content	String	Die detailierten Ausgaben jeder Mutation
eMut_evis_manually	String	Manueller Anzeige-String für „eVis"
eMut_mutation	String	Mutations-String der Datensätze
eMut_pdbid0	String	Die darzustellende PDB-ID (EP-ID) mit Index 0
eMut_pdbid1	Double	Die darzustellende PDB-ID (EP-ID) mit Index 1

5.8. servlet.ESearchServlet

Im Gegensatz zu den bisher beschriebenen Servlets, bietet das „ESearchServlet" eine größere Funktionalität an. Es verwaltet die Suche nach Energieprofilen.

Request-Attribut	Typ	Beschreibung
chain0	String	Die darzustellende Kette
esearchid	String	Ergebnis-ID einer fertigen Suche
esearchmail	String	eMail-Adresse für die Benachrichtigung
gapcost	Integer	Kosten für eine neue Lücke im „eAlign"
gapextend	Integer	Kosten für das erweitern einer Lücke im „eAlign"
hits	Integer	Anzahl der zu zeigenden Ergebnisse
method	String	Methode für „eAlign" (lokal oder global)
reset (Parameter)	Ohne Typ	Setzt alle eSearch-Session-Attribute zurück

Die berechneten Ergebnisse werden an die JSP „eSearch.jsp" weitergeleitet:

Session-Attribut	Typ	Beschreibung
eSearch	String	Schalter, zwischen der Eingabemaske und der aufbereiteten Ausgabe • null – Eingabemaske • „result" – Ergebnisse • „toomanyrequests" – Zu viele Anfragen • „sent" – Anfrage entgegengenommen • „enqueued" – Anfrage wird bearbeitet
eSearch_amount	Integer	Anzahl der durchsuchten Proteine
eSearch_amountRequests	Integer	Anzahl der gestellten Anfragen
eSearch_chain	String	Die darzustellende Kette
eSearch_gapcost	Integer	Kosten für eine neue Lücke im „eAlign"
eSearch_gapextend	Integer	Kosten für das erweitern einer Lücke im „eAlign"
eSearch_hitlist	ArrayList	Ergebnis-Liste der Proteine
eSearch_method	String	Methode für „eAlign" (lokal oder global)
eSearch_pdbid	String	Die darzustellende PDB-ID (EP-ID)
eSearch_queuePosition	Integer	Position innerhalb der Warteschlange

Das Servlet behandelt und verwaltet die gestellten Suchanfragen innerhalb einer Warteschlange.

```
queueSearch = new ESearchQueue();
queueThread = new Thread(queueSearch);

checkPending();

queueThread.start();
```

Um das System durch den Suchvorgang nicht zu sehr zu belasten, wird die Warteschlange in einem eigenen Thread verwaltet. Die Warteschlange, sowie der dazugehörige Thread werden innerhalb der init-Methode des Servlets initialisiert. Bevor der Thread gestartet wird, werden unfertige Anfragen neu in die Warteschlange eingetragen. Diese können beispielsweise durch einen Serverneustart vor ihrer Fertigstellung abgebrochen worden sein. Damit die Anfragen schnellstmöglich wieder in die Warteschlange eingereiht werden, wird das „ESearchServlet" als

einziges Servlet beim Start des Servers geladen und nicht beim ersten Aufruf. Dazu wird im Deployment Descriptor folgender Eintrag im Servlet-Tag hinzugefügt:

```
<load-on-startup>0</load-on-startup>
```

Eine Anfrage wird durch einen Ordner, mit der Session-ID und einer laufenden Nummer als Name (ESR-ID), sowie zwei Dateien beschrieben. Die erste Datei wird beim Erstellen der Anfrage erzeugt (Start.esr). Diese ESR-Datei beinhaltet alle relevanten Daten der Anfrage. Sobald die Bearbeitung erfolgreich durchgeführt wurde, wird die zweite ESR-Datei (Done.esr) erzeugt, welche neben den Eingabedaten auch die fertig berechneten Suchergebnisse beinhaltet. Die `checkPending`-Methode überprüft alle im ESR-Verzeichnis liegenden Ordner nach vorhandenen „Done.esr"-Dateien. Sollte ein Ordner nur die „Start.esr"-Datei beinhalten, so wird diese Anfrage erneut in die Warteschlange eingereiht.

Eine weitere Besonderheit des „ESearchServlet" ist der erweiterte Schalter („eSearch"-Attribut in der Session), der mehr als nur zwei Zustände unterstützt. Dabei hängt der gewählte Weg von den Eingabedaten des Nutzers ab.

Sobald ein Nutzer eine neue Anfrage erstellt, werden seine Eingabedaten ausgewertet. Sollte er innerhalb der aktuellen Session weniger als drei (steuerbar durch die Variable „MAX_AMOUNT_SESSION_REQUESTS" im „ESearchServlet") eingereihte Anfragen haben, so wird die aktuelle Anfrage in die Warteschlange eingereiht und eine Erfolgsmeldung („sent") angezeigt. Wenn mehr als drei wartende Anfragen vorhanden sind, so wird der Nutzer gebeten auf die Beendigung seiner aktuellen Anfragen zu warten („toomanyrequests"). Sobald eine Anfrage erfolgreich abgearbeitet wurde wird dem Nutzer eine eMail an die hinterlegte Adresse geschickt, über die er sich seine Ergebnisse anhand der erstellten ID betrachten kann („result"). Möchte der Nutzer eine ID abfragen, die in der Warteschlange eingereiht, aber noch nicht abgearbeitet wurde, wird er um etwas Geduld gebeten („enqueued").

5.8.1. ESearchQueue

Die Klasse „ESearchQueue" dient als Warteschlange für Anfragen, die über das „ESearchServlet" gestellt werden. Die Anfragen werden dabei in einer ArrayList aus „ESearchRequest"-Objekten verwaltet. Die einzige Möglichkeit auf die Warteschlange zuzugreifen ist über die Methode:

```
public synchronized int addRequest(ESearchRequest request);
```

Diese erzeugt den Ordner mit der ESR-ID und legt die „Start.esr"-Datei (serialisiert das `request`-Objekt) ab. Danach wird die Anfrage am Ende der Warteschlange eingereiht und die Größe der Warteschlange zurückgeliefert.

Innerhalb der run-Methode, welche beim Starten den Threads aufgerufen wird, wird die am Anfang der Warteschlange stehende Anfrage entnommen und die Bearbeitung gestartet. Dabei ist auf Grund der Thread-Problematik darauf zu achten, dass das Entnehmen und das Hinzufügen eines Eintrags in die Warteschlange synchronisiert werden muss. Wenn kein Eintrag in der Warteschlange vorhanden ist, wird eine 10 minutige Idle-Zeit eingeführt.

Die Suche nach ähnlichen Energieprofilen wird über den gesamten vorhandenen Datensatz ausgeführt. Dabei wird mit Hilfe von „eAlign" der zScore zwischen dem übergebenen Energieprofil und jedem im Datensatz liegenden Energieprofil berechnet. Die berechneten

Ergebnisse werden mit Hilfe einer ArrayList, bestehend aus „ESearchHit"-Objekten sortiert. Die Klasse „ESearchHit" ist dabei nur ein Container für die PDB-ID und den zScore. Durch das vom Nutzer eingestellte Attribut „`hits`" wird die Anzahl der zu zeigenden Ergebnisse beschränkt. Anhand dieser Anzahl werden Ergebnisse, deren zScore zu niedrig liegt nicht mit in der ArrayList gehalten.

Anschließend an die Bearbeitung wird das Ergebnis in Form der „Done.esr"-Datei abgespeichert und eine eMail zur Benachrichtigung des Nutzers versendet.

Die ESR-Dateien werden eine Woche lang gespeichert, bevor sie nach Ablauf dieser Zeit durch die Klasse „MySessionListener" gelöscht werden.

Das Versenden von eMails wird über die JavaMail-API von Oracle realisiert. Um den Mail-Service nutzen zu können, muss ein existierender eMail-Account angesprochen werden. Zu diesem Zweck wurde bei dem freien Anbieter web.de ein eMail-Account („epros.noreply@web.de") eingerichtet. Die JavaMail-API benötigt verschiedene Parameter, damit sie die Verbindung zum Mail-Server aufbauen kann. Die für web.de entsprechenden Einstellungen sind wie folgt:

```
Properties props = new Properties();

props.put("mail.transport.protocol", "smtp");
props.put("mail.smtp.host", "smtp.web.de");
props.put("mail.smtp.auth", "true");
props.put("mail.smtp.port", "587");
```

Weiterhin müssen die Login-Daten, Header-Daten und die Nachricht an sich übergeben werden, bevor eine eMail verschickt werden kann. Die Login-Daten sind im Quelltext der „`sendMail`"-Methode zu finden. Diese Variante der Speicherung sollte im Zuge der Weiterentwicklung jedoch aus Sicherheitsgründen noch geändert werden.

5.9. servlet.GetFile

Das Servlet „GetFile" unterscheidet sich von den restlichen Servlets, indem es keinen speziellen Tools zugeordnet ist, sondern verschiedene Tools nutzt. Es dient dabei der Bereitstellung und Erzeugung von Bildern, sowie Dateien. Es wird nicht über ein Formular angesprochen, sondern direkt über Parameter in der URL. Das „GetFile"-Servlet ist also über Parameter via GET-Methode ansprechbar. Die entsprechenden Parameter sind dabei in der folgenden Tabelle aufgeführt:

Parameter	Beschreibung
chain0	Die darzustellende Kette für Datensatz 0 (optional)
chain1	Die darzustellende Kette für Datensatz 1 (optional)
display	Regelt die Anzeige oder den Download der angeforderten Ressource
info	Zusätzliche Informationen
pdbid0	Die darzustellende PDB-ID (EP-ID) mit Index 0
pdbid1	Die darzustellende PDB-ID (EP-ID) mit Index 1
scale	Skalierungs-Modus für Bilder
type	Der Typ der Ressource • „ep" oder „ep2" – Liefert das Energieprofil im Textformat • „dotplot" – Liefert einen Dotplot • „mutation" – Liefert ein Mutations-Diagramm

Je nach Art des angeforderten Typs (type) werden nur bestimmte Parameter benötigt. Die Angabe einer Kette ist für alle Anfragen bezüglich einer PDB-ID optional. Das Anfordern von Energieprofilen (type hat den Wert „ep" oder „ep2") funktioniert nach diesen Mustern:

```
// Download eines Energieprofils im Textformat
GetFile?type=ep&pdbid0=1A0S                    // 1A0S komplett
GetFile?type=ep&pdbid0=1A0S&chain0=Q           // Kette Q von 1A0S

// Anzeige eines Energieprofils im Textformat
GetFile?type=ep2&pdbid0=1DSL&display
GetFile?type=ep&pdbid0=1A0S&chain0=Q&display
```

Die Möglichkeiten einen Dotplot anzufordern (type hat den Wert „dotplot") ist mit diesen Kombinationen möglich:

```
// Anzeigen eines Dotplots (geeignet für separate Fenster)
GetFile?type=dotplot&pdbid0=1DSL&pdbid1=1AMM&display
GetFile?type=dotplot&pdbid0=1A0S&pdbid1=1AMM&chain0=Q&display

// Einbinden eines Dotplot-Bildes (geeignet für img-Tags)
GetFile?type=dotplot&pdbid0=1DSL&pdbid1=1AMM&scale   // Skaliert
GetFile?type=dotplot&pdbid0=1DSL&pdbid1=1AMM         // Originalgröße
GetFile?type=dotplot&pdbid0=1A0S&pdbid1=1AMM&chain0=Q
```

Ein Mutationsdiagramm kann über die folgende Anfrage abgerufen werden:

```
// Einbinden eins Mutationsdiagramms (geeignet für img-Tags)
GetFile?type=mutation&pdbid0=1FFA&pdbid1=1FFB&info=0
GetFile?type=mutation&pdbid0=3AME&pdbid1=9AME&info=2
```

Das Servlet wertet dabei die übergeben Parameter aus und erzeugt anhand der Informationen der einzelnen Tools, die angesprochen werden, die entsprechenden Bilder oder Dateien.

6. JSP

Die JSP-Seiten dienen der Eingabe der Nutzerdaten, sowie der Darstellung der berechneten Ergebnisse. Dabei kann „eProS" in zwei verschiedene Arten von JSP-Seiten getrennt werden. Zum Einen sind das die normalen JSPs, welche hauptsächlich HTML-Code enthalten. Zum Anderen sind das die JSP-Seiten für die Tools, welche mit Java-Code durchzogen sind.

Um Java-Code in JSP-Seiten einzubringen muss eine zusätzliche Kopfzeile in die JSP eingetragen werden:

```
<%@page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
```

Diese „page"-Direktive erleichtert es dem Compiler die Inhalte der JSP-Seite zu verstehen und verweist über das Attribut „language" mit dem Wert „java" darauf, dass in dieser Seite Java-Code eingebettet ist.

Es gibt vier verschiedene JSP-Anweisungen, die „eProS" auf seinen JSP-Seiten verwendet. Die erste Anweisung ist die „include"-Direktive.

```
<%@include file="fileToInclude" %>
```

Mit dieser Direktive wird es möglich, den kompletten Inhalt einer anderen Datei (`fileToInclude`) in die JSP einzubinden. Die Einbindung des Inhalts geschieht, bevor die JSP in ein Servlet transformiert wird. Die Datei, die eingebunden wird, wird dabei durch das Attribut "file" angegeben.

Die zweite Anweisung, die „include"-Standardaktion, ähnelt der „include"-Direktiven sehr stark.

```
<jsp:include page="fileToInclude">
      <!-- optinaler Parameterbereich -->
      <jsp:param name="ParamName" value="ParamValue" />
</jsp:include>
```

Die Unterschiede zwischen diesen beiden Varianten ist einerseits, die Übergabe-Möglichkeit von Parametern innerhalb der Standardaktion. Andererseits die Art, wie der Inhalt in die Seite eingebunden wird. Bei der Direktive wird der Inhalt der angegebenen Datei vor dem Übersetzen in die JSP Seite eingefügt, wobei die Standardaktion den Inhalt, den die Datei zurück liefert zur Laufzeit in die Seite einfügt.

Die dritte verwendete Anweisung ist ein JSP-Ausdruck. Als Ausdruck wird alles gewertet, was einen primitiven Wert oder ein Objekt zurück gibt. Um diesen in die JSP einzubinden wird folgender Syntax benötigt:

```
<%= request.getParameter("ParamName") %>
```

Die vierte JSP-Anweisung, die auf den JSP-Seiten von „eProS" Anwendung findet, ist das Skriptlet. Skriplets können beliebigen Java-Code enthalten, sollten jedoch mit Vorsicht eingesetzt werden, da sie die JSP-Seiten schnell unübersichtlich machen können. Um Java-Code in Form von Skriplets in eine JSP einzubinden muss folgender Syntax verwendet werden:

```
<%
      // Beliebiger Java-Code
%>
```

6.1. Die Nicht-Tool-Seiten

Diese JSP-Seiten dienen der Anzeige der normalen statischen Inhalte, diese sind im Pfad „/WebContent/jsp/page/" zu finden:

- about.jsp
- contact.jsp
- home.jsp
- index.jsp
- introduction.jsp
- navigation.jsp
- reference.jsp
- statistics.jsp

Die Seiten liefern das Grundgerüst für „eProS". In ihnen finden sich alle Hintergrundinformationen und Inhalte wieder, die für die Anwendungen und Tools nicht direkt notwendig sind, aber zu ihrer Erstellung beigetragen haben (siehe statistics.jsp).

Die „index.jsp" ist die Startseite von „eProS" und besteht aus zwei Frames. Der linke Frame wird mit der Navigation („navigation.jsp") und der rechte Teil mit der Newsseite („home.jsp") geladen. Dabei kommt die Sonderstellung der „index.jsp" zum tragen. Sie prüft, ob einen Parameter mit dem Name „id" übergeben wurde, falls dies der Fall ist, wird nicht mit der „home.jsp" als Startseite begonnen, sondern der Wert des Parameters gleich an die Tool-Seite für „eSearch" durchgereicht.

Die restlichen Seiten werden über die Navigation angesprochen und im rechten Frame der „index.jsp" dargestellt.

6.2. Die Tool-Seiten

Die zweite Art von JSP-Seiten in „eProS" sind die Tool-Seiten. Diese stehen im direkten Zusammenhang mit den entsprechenden Servlets und bieten angepasste Eingabemöglichkeiten, sowie Ausgabeformate an.

Alle Tool-Seiten, die „Jmol" oder „eVis" einbinden und damit PDB- oder EP-Dateien benutzen, verwenden folgendes Standard-Skriptlet oder eine erweiterte, modifizierte Variante:

```
<%
String strPDBID   = (String)session.getAttribute("eXXX_pdbid");
strPDBID          = strPDBID.toUpperCase()
String strChain   = (String)session.getAttribute("eXXX_chain");

String strPDBFile = session.getAttribute("urlPDB") +
                    session.getId() + "/" + strPDBID + ".pdb";

try { new URL(strPDBFile).openStream(); }
catch (Exception ex) {
strPDBFile        = (String)session.getAttribute("urlPDB") +
                    strPDBID + ".pdb"; }

try { new URL(strPDBFile).openStream(); }
catch (Exception ex) {
strPDBFile        = null; }

// Die gleiche Abfrage erfolgt anschließend für das strEPFile und alle
// anderen übergebenen PDB- und EP-Attribute

%>
```

Dieses Skriptlet liest alle PDB-IDs und die dazugehörigen Ketten aus den Session-Attributen aus und sucht nach der PDB-, sowie der EP-Datei. Als erstes wird dazu im Session-Ordner geschaut, so können hochgeladene Dateien vor denen im Datensatz bevorzugt werden. Falls die Datei nicht im Session-Ordner vorhanden ist, wird sie im Datensatz angenommen. Wenn sie auch nicht im Datenordner vorhanden ist, wird sie mit `null` referenziert.

Neben diesem Standard-Skriplet, was nahezu alle JSP-Seiten nutzen, entspricht auch der Aufbau der JSPs dem gleichen Muster:

```
<html>
<head>
      <%@include file="/jsp/include/toolsHeader.jsp" %>
</head>

<body>

<%@include file="/jsp/include/toolsPleaseWait.jsp" %>

<!-- Schalter -->
<% if (session.getAttribute("eXXX") == null) { %>

      <!-- Eingabemaske -->

<% } else { %>

      <!-- Standard-Skriptlet -->
      <a href="/Epros/EXXXServlet?reset">Back to input-mask</a>
      <!-- Inhalt -->

<% } %>

</body>
</html>
```

An diesem Muster ist zu erkennen, dass alle JSP-Seiten den selben Header („/jsp/include/toolsHeader.jsp“) laden. Damit verbunden sind Stylesheets, sowie JavaScripts. Die erste Anweisung im Body-Tag lädt das Wartefenster in die JSP, welches angezeigt wird, sobald ein Formular abgeschickt wird. Es zeigt dem Nutzer an, dass ein zeitintensiver Prozess gestartet wurde und der Browser sich nicht aufgehangen hat.

Anschließend ist der Schalter für die jeweiligen Anzeigen zu finden. Demnach wird je nach Status des Schalters ein anderer Teil der JSP-Seite dargestellt. Für die „eSearch.jsp“ werden noch weitere Verzweigungen („else if“-Bereiche) eingefügt. Die Zeile zwischen dem Standard-Skriptlet und dem Inhalt, ruft das entsprechende Servlet mit einem „reset“-Parameter auf, um zurück zur Eingabemaske zu gelangen.

Die Eingabemaske besteht ebenfalls aus festen Bestandteilen. Dazu zählt zum einen das Feld für die Dateneingabe, zum anderen wahlweise ein Optionen-Feld, sowie die Möglichkeit die Session-Daten zu löschen und die Eingaben an das Servlet zu übermitteln. Das Eingabefeld für die Daten wird dabei über die „include“-Standardaktion und der Übergabe eines Index als Identifier eingebunden:

```
<jsp:include page="/jsp/include/input.jsp">
      <jsp:param name="index" value="0" />
</jsp:include>
```

Ein weiterer Parameter, die der „input.jsp" übergeben werden kann, beeinflusst das Aussehen, indem er zwischen der normalen Eingabe-Maske und dem Sequenz-Feld wechselt:

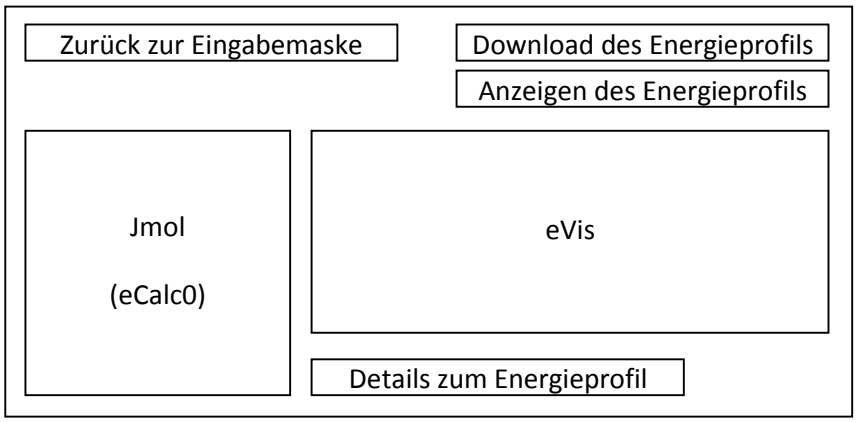
```
<jsp:param name="sequence" value="visible" />  // Nur Sequenz-Feld
<jsp:param name="sequence" value="include" />  // Mit Sequenz-Feld
```

Die zusätzlichen Optionen müssen manuell bearbeitet werden. Dabei ist es wichtig bei der Namensgebung der Elemente eine abschließende „0" als Index zu formulieren. Die zusätzlichen Optionen müssen dann im „MyStandardServlet" ausgelesen werden.

Dieser Aufbau sorgt für ein einheitliches Aussehen der einzelnen Tool-Seiten und ermöglicht eine intuitive Nutzung.

6.2.1. eCalc

Der Aufbau des Inhaltes der „eCalc.jsp":

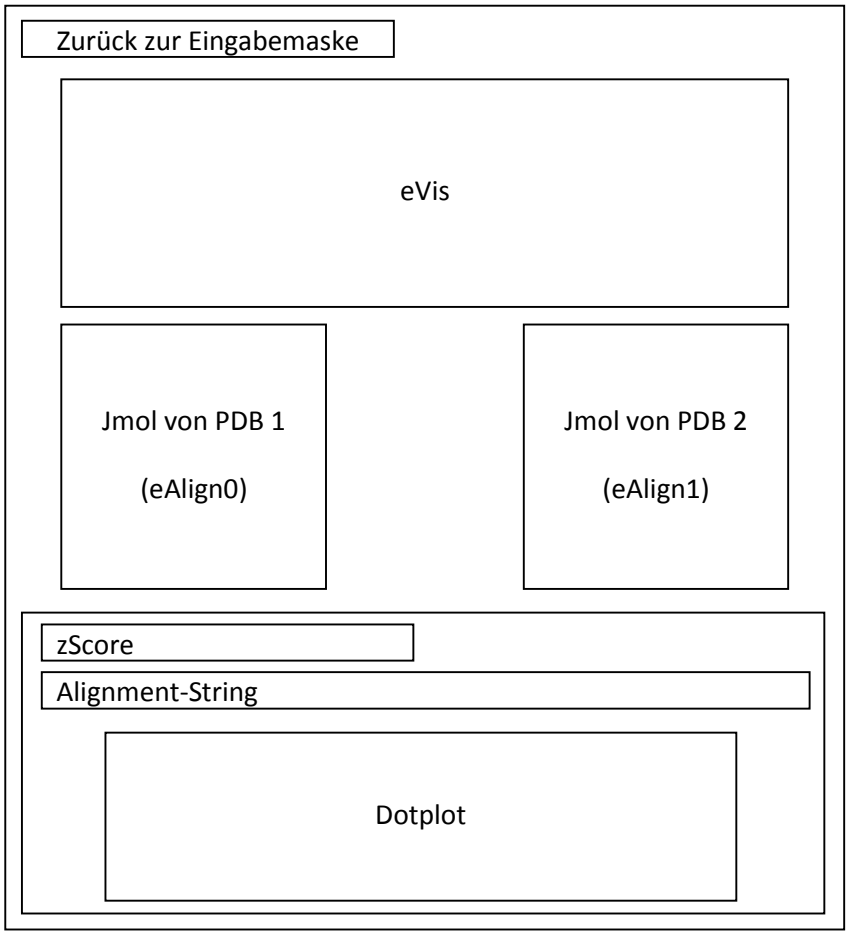


An „eVis" übergebene Parameter:

```
<param name="suffix0" value="eCalc0" />              // Name von Jmol
<param name="eps0" value="<%= strEPPath %>" />       // EPS-Datei
<param name="chain0" value="<%= strChain %>" />      // Kette
```

6.2.2. eAlign

Der Aufbau des Inhaltes der „eAlign.jsp“:



An „eVis“ übergebene Parameter:

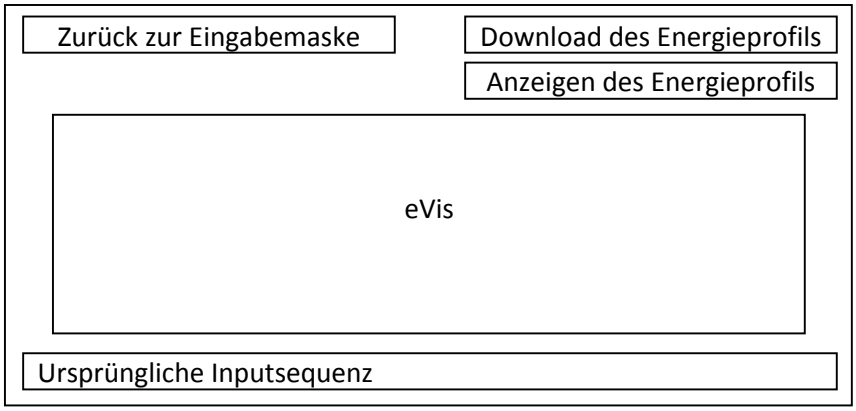
```
// Informationen über den ersten Datensatz
<param name="eps0" value="<%= strEPFile0 %>" />
<param name="chain0" value="<%= strChain0 %>" />
<param name="suffix0" value="eAlign0" />

// Informationen über den zweiten Datensatz
<param name="eps1" value="<%= strEPFile1 %>" />
<param name="chain1" value="<%= strChain1 %>" />
<param name="suffix1" value="eAlign1" />

// Der alignierte Energie-String
<param name="align"
       value="<%= session.getAttribute("eAlign_evis_alignment") %>">
```

6.2.3. eGor

Der Aufbau des Inhaltes der „eGor.jsp“:

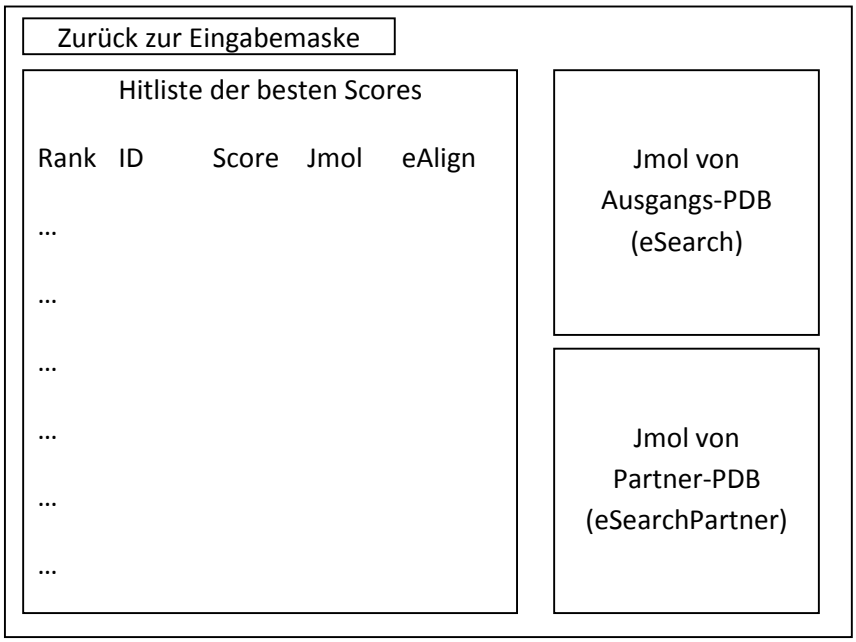


An „eVis“ übergebene Parameter:

```
<param name="eps0" value="<%= strEPFile %>" />
```

6.2.4. eSearch

Der Aufbau des Inhaltes der „eSearch.jsp“:

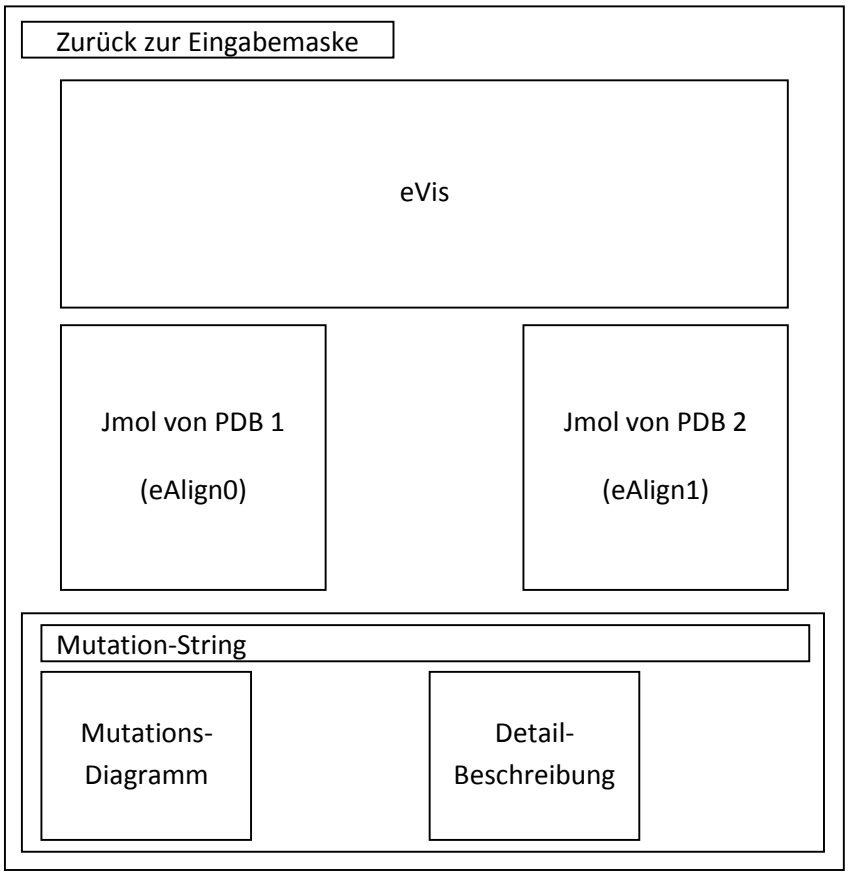


Über die Links in der Spalte „Jmol“ können die jeweiligen Einträge der Hitliste im Jmol-Applet (eSearchPartner) visualisiert werden. Die Links in der Spalte „eAlign“ nutzen die entsprechenden Daten um das gesuchte Energieprofil und den jeweiligen Eintrag der Hitliste mit „eAlign“ darzustellen. Dazu wird ein verstecktes Formular (hidden-Attribute) an das „EAlignServlet“ gesendet.

Die restlichen Seiten, die „eSearch“ bereitstellt, besitzen keinen speziellen Aufbau, sie informieren den Nutzer mit kurzen Textabschnitten über das entsprechende Thema.

6.2.5. eMut

Der Aufbau des Inhaltes der „eMut.jsp“:



An „eVis“ übergebene Parameter:

```
<param name="mutation_mode" value="true" />
<param name="manually0"
       value="<%= session.getAttribute("eMut_evis_manually") %>" />
<param name="suffix0" value="eMut0" />
<param name="suffix1" value="eMut1" />
```

Die Parameter, die an „eVis“ übergeben werden, beschreiben einen Datensatz (`manually0`), es werden aber zwei „Jmol“-Applets über die Suffixe angemeldet. In diesem Fall werden die Daten von „eVis“ an beide „Jmol“-Applets geschickt. Markierungen aus den Applets werden nur auf den einen Datensatz bezogen, somit bleibt die Funktionalität und Kommunikation erhalten.

6.3. Die Fehler-Seiten

Der Deployment Descriptor ermöglicht es auftretende Fehler abzufangen und auf eigene Fehlerseiten umzuleiten. Somit können diese Seiten an das eigene Design angepasst und die Behandlung der jeweiligen Fehler selbst gesteuert werden. Der entsprechende Verweis im Deployment Descriptor kann so eingetragen werden:

```
// Umleiten von HTTP-Status-Codes (Error-Codes)
<error-page>
      <error-code>404</error-code>
      <location>/jsp/error/error404.jsp</location>
</error-page>

// Umleiten von geworfenen Exceptions
<error-page>
      <exception-type>java.lang.Throwable</exception-type>
      <location>/jsp/error/error.jsp</location>
</error-page>
```

Die erste Möglichkeit erlaubt es direkt HTTP-Status-Codes abzufangen. Mit der zweiten Variante können geworfene Exceptions umgeleitet werden. Diese beiden Angaben werden in „eProS", so wie sie als Beispiel gegeben sind, verwendet. Der Statuscode 404 für die „Page not found"-Meldung wird an das Design der Seite angepasst. Alle anderen auftretenden Fehler werden über die zweite Variante abgefangen. Die „error.jsp" bereitet den Fehler dabei auf, damit es möglich ist gezielter nach der Ursache zu suchen.

An dieser Stelle sollte in einer späteren Phase der Entwicklung ein Error-Listener implementiert werden, der eine eMail mit der Fehlerbeschreibung an den entsprechenden Administrator sendet.

7. Ausblick

„eProS" bildet mit seinen jetzigen Tools eine solide Grundlage. Im Laufe der weiteren Forschung und Entwicklung können die derzeitigen Tools noch optimiert werden und neue Tools einen Platz auf dem Server finden. Der lokale Datensatz kann um einige Proteine erweitert werden um die Performance erheblich zu steigern und die Wartezeiten zu minimieren. Es existieren noch einige sicherheitsrelevante Bereiche, in denen man aktiv werden müsste.

Der Produktionsserver ist eine virtuelle Maschine, die über einen Proxy angesprochen wird. Damit verbunden sind einige Einbußen in der Perfomance. Ein eigener leistungsstarker Server würde diesem Einbruch entgegenwirken. Das Thema sollte aufgegriffen werden, sobald abschätzbar ist, wie viele Nutzer, wie viel Traffic verursachen. Auf Grund der vergleichsweise leistungsschwachen VM, wird die Suche nach Energieprofilen sequentiell durchgeführt. Mit einem leistungsstärkeren Server könnte dieser Prozess parallelisiert werden, was eine erhebliche Zeitersparniss einbringen würde.

Um die Suche noch effektiver zu gestalten, sollte „eSearch" in eine heuristische Suche umgewandelt werden. Diese könnte die derzeitige, sehr zeitintensive Suche, ablösen. Das Speichern der Suchergebnisse auf dem Server kann umgangen werden, indem die Ergebnisse in Form eines PDFs aufbereitet und an den Nutzer gesendet werden, anstatt diesem eine ID zukommen zu lassen.

Das Vorhersagen von Sekundärstrukturen, sowie speziellen Regionen und Motiven anhand von Energieprofilen würde eine Weiterentwicklung darstellen. Über statistische Auswertungen und Trends könnten damit die Energieprofile noch genauer spezifiziert werden. Die Analyse von Protein-Familien spielt dabei wohl eine nicht ganz unbedeutende Rolle.

Die einzelnen Tools sind derzeit für den Nutzer so zugänglich, dass angeforderte Ergebnisse bestehen bleiben. Der nächste Schritt wäre, dass Veränderungen und vor allem Interaktionen, die der Nutzer an den Ergebnisseiten vorgenommen hat erhalten bleiben.

Literaturverzeichnis

[1] Heinke, Florian: Energieprofilbasierende Analysemethoden von Proteinfamilien, Bachelorarbeit

[2] Dressel, Frank: Sequenz, Energie, Struktur - Untersuchungen zur Beziehung zwischen Primär- und Tertiärstruktur in globulären und Membran-Proteinen, Dissertationsschrift

[3] Schneider, Uwe; Werner, Dieter: Taschenbuch der Informatik. – 6. Aufl. – Mittweida: Hanser Fachbuch, 2007

[4] Krüger, Guido; Stark, Thomas: Handbuch der Java-Programmierung. – 6. Aufl. – München: Addison-Wesley, 2009

[5] Jmol Interactive Script Documentation. URL: <http://chemapps.stolaf.edu/jmol/docs/>

[6] Jmol Wiki. URL: <http://wiki.jmol.org/index.php/Main_Page>

[7] Java Tips – Introduction to JavaServlets with Eclipse. URL: <http://www.java-tips.org/java-tutorials/tutorials/introduction-to-java-servlets-with-eclipse.html>

[8] A Tutorial on Java Servlets ans Java Server Pages (JSP). URL: <http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/>

[9] Apache Tomcat 6.0 – Documentation. URL: <http://tomcat.apache.org/tomcat-6.0-doc/index.html>

[10] Java Server Pages: Servlet und JSP Server Tomcat. URL: <http://www.java-server.net/jsp_server.html>

[11] The Ultimate JSP Tabs!. URL: <http://ditchnet.org/tabs/>

[12] JavaServer Pages (JSP) and JSTL – jsp and JPanel problem. URL: <http://forums.sun.com/thread.jspa?threadID=600587>

[13] Applet in ein JSP einbinden – java-forum.org. URL: <http://www.java-forum.org/allgemeines-ee/102914-applet-jsp-einbinden.html#post655740>

[14] Call javascript from a Java applet – Real's Java How-to. URL: <http://www.rgagnon.com/javadetails/java-0172.html>

[15] Technische Kurzdokumentationen Torsten Horn. URL: <http://www.torsten-horn.de/techdocs/index.htm#JSP>

[16] FileUpload - Home. URL: <http://commons.apache.org/fileupload/>

[17] FileUpload – Using FileUpload. URL: <http://commons.apache.org/fileupload/using.html>

[18] Uploading Files with Beans – O'Reilly Media. URL: <http://tim.oreilly.com/pub/a/onjava/2001/04/05/upload.html>

[19] Mail aus Servlet verschicken? @ tutorials.de: Tutorials, Forum & Hilfe. URL: <http://www.tutorials.de/java/228224-mails-aus-servlet-verschicken.html>

Verwendete Tools und Programme

Die verwendete Programmiersprache mit der die Programme geschrieben wurden, war ausschließlich Java. Als Java-Version wurde Java 6 Update 12 genutzt.

Die Entwicklungsumgebung war Eclipse Helios.

Die Erstellung der UML-Diagramme erfolgte mit Hilfe von „astah* community" (http://astah.change-vision.com/en/product/astah-community.html) früher als „Jude" bekannt.

Zur Erstellung des Struktogramms wurde der „NSD Editor" herangezogen (http://www.tinohempel.de/info/info/info-cd/index2.htm).

Der Apache Tomcat in der Version 6.0.26. wurde als lokaler Testserver genutzt.

Der Zugriff auf den Produktionsserver erfolgte über den Cisco VPN Client 5.0.02.0090, sowie Putty 0.60 und FileZilla 3.3.4.1.

Erklärung zur selbständigen Anfertigung der Arbeit

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Literatur angefertigt habe.

Seifersdorf, den 22.08.2010 Stefan Schildbach